# Algorithms for Massively Parallel, Event-Based Hardware

Dissertation zur Erlangung der naturwissenschaftlichen Doktorwürde

(Dr. sc. nat.)

vorgelegt der

Mathematisch-naturwissenschaftlichen Fakultät

der

**Universität Zürich**

von

**Lorenz Kaspar Müller**

von

Winterthur, ZH

Promotionskommittee

Prof. Dr. Giacomo Indivieri (Vorsitz)

Dr. Michael Pfeiffer

Dr. Matthew Cook

Zürich 2015

# Acknowledgments

Many people have helped me in a great variety of ways in the making of this thesis. I would like to thank the people of INI, who collectively made working a pleasure.

In particular I would like to thank my supervisor Giacomo Indiveri, who gave me the opportunity to do this work and supported me with many, always helpful suggestions along the way.

I am grateful to Matthew Cook and Michael Pfeiffer for their much needed feedback on early versions of this thesis.

I am also very thankful to Hesham Mostafa, who collaborated with me on most of the topics I describe in this thesis, without whom this thesis would surely have taken a very different shape and who taught me a lot about how to do good and relevant science.

Last but not least I would like to thank my family for their continuous support and my wife, Liv, without whose advice I would not have dared applying for this position, who moved to Switzerland so that I could pursue this work and who helped me stay motivated through all this time.

# Abstract

The future of computing is uncertain: Attempting to keep up with the promise of exponential growth of computational power over time implicit in Moore's Law, chip manufacturers have in recent years had to place increasingly many processing cores on single chips. At the same time some domains of high performance computing have adopted graphics processing units (GPUs) as their substrate of choice. GPUs more genuinely embrace parallelism as a design philosophy by combining hundreds of parallel cores with high memory bandwidth. Massively parallel, event-based, 'neuromorphic' hardware take parallelism to the extreme: Memory and processing are fully distributed and if centralized structures exist it is only to route very short messages (events) between processing elements.

Additionally the communication infrastructure of these neuromorphic platforms departs from the von-Neumann architecture and the architecture of GPUs: Instead of coupling the parallel cores over a globally accessible state stored in a distant memory, the communication is reduced to short and sparse messages (events) exchanged directly between computational nodes. This approach has the potential to overcome performance limitations imposed by the 'von-Neumann bottleneck', i.e. the need to pass information back and forth between memory and processor.

Numerous neuromorphic platforms exists today, and offer in principle high performance at a low power consumption. However in contrast to von-Neumann architectures for which algorithms have been developed for over fifty years, there is a distinct lack of algorithms for massively parallel, event-based hardware.

In the field of theoretical neuroscience many such algorithms have been described, but for a somewhat different substrate: Biological neurons. When physical implementability is discussed in these descriptions, it refers to the plausibility that the algorithm in question is used in some form in the brain. This 'physical implementability' however does not always relate to the physical implementability on neuromorphic silicon chips, probably because it is difficult to *limit* the biologically plausible, while on silicon it is difficult to *extend* the repertoire of available functionality.

In this thesis I develop several algorithms for the application to massively parallel,

event-based, electronic hardware. A hallmark of algorithms well suited to this computing style seems to be their decomposition into nearly independent sub-parts that exchange only sparse information with each other. This is particularly the case for the stochastic local search and related algorithms that I focus on in this thesis.

First I construct an abstract continuous-time model of neural network dynamics and show that it behaves like a discrete-time Markov chain Monte Carlo sampler; a ubiquitously used method for sampling from probability distributions that are difficult to access directly. I examine analytically and in simulation the relationship of this model to Artificial Neural Networks, a class of powerful generative model learners and classification algorithms. I further study a variant of this model based on coupled, mismatched oscillators that can be implemented in massively parallel, event-based hardware efficiently.

I extend the approach of constructing Markov chain Monte Carlo samplers on this hardware to the related and more general stochastic local search algorithms. These algorithms are applicable to solving constraint satisfaction problems, rather than sampling. I show that an event-based, massively parallel network can efficiently solve boolean satisfiability problems.

# Zusammenfassung

Die Zukunft der Informationsverarbeitung ist ungewiss: Im Unterfangen dem Versprechen exponentiellen Wachstums der Rechenleistung, das Moore's Law impliziert, gerecht zu werden, sehen sich Chiphersteller in den letzten Jahren gezwungen immer mehr CPU Kerne auf einzelnen Chips zu platzieren. Zugleich haben sich einige Bereiche der Hochleistungsinformationsverarbeitung den Grafikkarten (GPUs) als bevorzugte Plattform zugewandt. GPUs setzen Parallelismus as Designphilosophie direkter um, indem sie hunderte paraller Kerne mit hoher Speicherbandbreite kombinieren. Massiv parallele, eventbasierte 'neuromorphe' Hardware führt den Parallelismus zum logischen Schluss: Speicher und Verarbeitung sind vollständig dezentralisiert und wenn zentrale Strukturen existieren, dann nur, um sehr kurze Nachrichten (events) zwischen den Verarbeitungselementen zu übermitteln.

Zusätzlich unterscheidet sich die Kommunikationsinfrastruktur dieser neuromorphen Plattformen von der von-Neumann Architektur und der Architektur der GPUs: Statt dass parallele Kerne über einen global zugänglichen Zustand, der in einem entfernt gelegenen Speicher liegt, gekoppelt werden, ist die Kommunikation beschränkt auf den direkten Austausch kurzer und seltener Nachrichten zwischen den Verarbeitungselementen. Dieser Ansatz hat das Potential Limitationen, die durch den von-Neumann-Flaschenhals (d.h. die Notwendigkeit des Austauschs von Information zwischen Prozessor und Speicher) zu Stande kommen, zu überwinden.

Zahlreiche neuromorphe Plattformen existieren bereits und bieten prinzipiell hohe Leistung bei tiefem Verbrauch. Im Gegensatz zu von-Neumann Architekturen, für welche seit Jahrzehnten Algorithmen entwickelt werden, besteht jedoch für massiv parallele, eventbasierte Hardware ein deutlicher Mangel an geeigneten Algorithmen.

Im Gebiet der theoretischen Neurowissenschaften sind einige solcher Algorithmen entwickelt worden, allerdings für ein anderes Substrat: Biologische Neuronen. Wenn die physikalische Umsetzbarkeit in diesen Beschreibungen diskutiert wird, betrifft dies die Plausibilität, dass der vorgeschlagene Algorithmus im Gehirn umgesetzt wird. Diese Art von physikalischer Umsetzbarkeit ist jedoch nicht immer Deckungsgleich mit der physikalischen Umsetzbarkeit auf neuromorphen Halbleiterchips, wahrscheinlich weil es einerseits

schwierig ist, das biologisch Plausible *einzuschränken*, während es andererseits schwierig ist das auf Halbleiterchips umsetzbare Repertoire *zu erweitern*.

In dieser Dissertation entwickle ich mehrere Algorithmen für die Anwendung auf massiv paralleler, eventbasierter Hardware. Ein Kennzeichen der Algorithmen, die sich gut für diesen Informationsverarbeitungsstil eignen, scheint die Zerlegbarkeit in fast unabhängige Substrukturen zu sein, die nur geringen Informationsaustausch benötigen. Dies trifft besonders auf die stochastische, lokale Suche und verwandte Algorithmen zu, auf die ich mich in dieser Dissertation konzentriere.

Zuerst konstruiere ich ein abstraktes Model der Dynamik neuraler Netze in kontinuierlicher Zeit und zeige, dass sich dieses wie ein Markov Chain Monte Carlo Sampler (eine weit verbreitete Methode des Samplens von schwer darstellbaren Wahrscheinlichkeitsverteilungen) in diskreter Zeit verhält. Ich untersuche die Beziehung zwischen diesem Model und künstlichen neuralen Netzen (einer Klasse leistungsfähiger Klassifizierungsalgorithmen) analytisch und in Simulation. Weiter untersuche ich eine Variante dieses Models basierend auf gekoppelten, fehlangepassten Oszillatoren, das sich effizient auf massiv paralleler, eventbasierter Hardware umsetzen lässt.

Ich erweitere danach den Ansatz der Konstruktion eines Markov chain Monte Carlo Samplers für solche Hardware auf die allgemeineren stochastichen, lokalen Suchalgorithmen. Diese Algorithmen sind zur Lösung von Bedingungserfüllungsproblemen geeinget. Ich zeige, dass ein massiv paralleles, eventbasiertes Netzwerk Erfüllbarkeitsprobleme der Aussagenlogik effizient lösen kann.

# Disclaimer

I hereby declare that the work in this thesis is that of the candidate alone, except where indicated in the text, and as described below.

Chapter 3 is partially based on the papers "Rhythmic inhibition allows neural networks to search for maximally consistent states" by Hesham Mostafa, Lorenz K. Müller and Giacomo Indiveri [MMI15b] and "Recurrent Networks of coupled WTA-oscillators for solving constraint satisfaction problems" by Hesham Mostafa, Lorenz K. Müller and Giacomo Indiveri [MMI13].

Chapter 4 is partly based on the paper "Rhythmic inhibition allows neural networks to search for maximally consistent states" by Hesham Mostafa, Lorenz K. Müller and Giacomo Indiveri [MMI15b].

Chapter 5 is partially based on the paper "Rounding Methods for Neural Networks with Low Resolution Synaptic Weights" by Lorenz Müller and Giacomo Indiveri [MI15].

Chapter 6 is partially based on the paper "Recurrent Networks of coupled WTA-oscillators for solving constraint satisfaction problems" by Hesham Mostafa, Lorenz K. Müller and Giacomo Indiveri [MMI13] and the paper "An event-based architecture for solving constraint satisfaction problems" by Hesham Mostafa, Lorenz K. Müller and Giacomo Indiveri [MMI15a].

Appendix B is partially based on the paper "Rhythmic inhibition allows neural networks to search for maximally consistent states" by Hesham Mostafa, Lorenz K. Müller and Giacomo Indiveri [MMI15b].

In the concerned sections (that are specified at the beginning of the chapters) other authors of those publications contributed through discussions and by editing text.

# Contents

Chapter 1

# Introduction

## 1.1 Motivation

Neuromorphic engineering, a term coined in the 1980's by Carver Mead, is the discipline of constructing computing devices and sensors whose architecture or processing style takes inspiration from biological neural systems [Mea89]. Some of the typical organizational principles that are used in neuromorphic systems follow. Neuromorphic systems are often *decentralized*, their components operate without any knowledge of large parts of the whole system. In particular there is often no global clock signal, so that different components operate *asynchronously*. This has the benefit that state changes are only required when computationally relevant, resulting in a *lower power consumption*. Memory and computing are *co-localized* in large numbers of units that communicate with *point events* (also called spikes); the flow of information between them is *massively parallel*.

While neuromorphic engineering has developed mainly with the goal of facilitating neuroscientific simulations, hardware with massively parallel organization is interesting independent of neuroscience, because it may be better suited for solving certain types of problems than standard von Neumann architectures. So-called 'best-match' problems [RM86] are theorized to be especially well suited for a massively parallel computing style.

In the case of machine learning this has been confirmed by a group of intrinsically parallel algorithms called Artificial Neural Networks. These algorithms are state-of-the-art in numerous classification tasks [Sch15]. However, although they were (at least in part) conceived to model perceptual inference in the brain [HS83], practically useful formulations are adapted either to server farms, GPUs or PCs as the underlying computational substrate. Formulating ANNs for these substrates leads to conceptual simplifications at the cost of computational efficiency (sparsity in activation and in the connection matrix cannot be utilized).

For neuromorphic systems to make good on the claim of computational usefulness (beyond neuroscientific simulations) there is a need for algorithms formulated for hardware

that follows the organizational principles of neuromorphic hardware and is physically well implementable. The aim of this thesis is to provide a few such algorithms.

## 1.2 Key Concepts

In this section I will give a short introduction to several useful concepts that should help readers from different fields understand various parts of this thesis more easily. I cannot give a general introduction to all relevant topics and the different descriptions require varying amounts of background knowledge. The most useful way of reading this section is to read the paragraphs on unfamiliar topics and skip familiar topics and topics whose explanations start from unfamiliar ground.

**Event-based Algorithms** Event- (or Spike-) based algorithms are algorithms formulated on a computational substrate in which networks of simple processing entities solve a problem collaboratively by exchanging information in the form of small information packets (events). A hallmark of this style of computation is 'on-demand' resource consumption: Parts of the network that are irrelevant for a particular computation are idle and do not consume resources.

Note that some other authors may use the term to refer to algorithms that deal with data composed of events on a conventional computational substrate.

**Neural Networks** 'Neural Networks' is an overloaded term. It has a general meaning (sets of interconnected neurons) and in this meaning it can be used in contexts from neuroanatomy to theoretical neuroscience. In computer science 'Neural Networks' has a special meaning and describes a class of machine-learning algorithms that are historically based on models of biological neurons and the brain. All these algorithms can be thought of in terms of simple units (neurons) that sum input (integration of current on the membrane capacitance), which comes from other units over weighted connections (synapses). Furthermore there is a desired activity for some of the units and a systematic method to change the connection weights, to achieve this activity. Notably some of these algorithms achieve state-of-the-art performance in tasks like image recognition or language modelling [Sch15].

To disambiguate, I will use the term 'artificial neural network' (ANN) when referring to the machine learning algorithms.

**Deep Learning** Deep learning is a non-specific term to refer to several ANN algorithms, if the ANN has more than one hidden layer. Intuitively one could say that each layer in an ANN performs an abstraction on its input data; a deep network thus can represent complex concepts as hierarchies of abstractions. For example in an image recognition task

a first layer can represent edges, a higher layer compositions of edges and an even higher layer entire object shapes.

Deep ANNs are the type of ANNs that have achieved state-of-the art performance in various classification tasks.[Sch15]

**Winner-Take-All** The Winner-Take-All is a particularly well-studied small circuit of neural populations. It is biologically relevant because it has been proposed as a candidate canonical micro-circuit in the neocortex [DM07]. Computationally it is of interest, because it has been shown to be a universal function approximator [Maa00] and the closely related [SMKGS13] max-out activation function is the building block of a state-of-the-art image classification network [GWFMCB13a] among other reasons

**Attractor States** If a dynamical system's trajectory passes sufficiently closely by an attractor state, the system will remain near that state indefinitely [Mil04]. Attractor states are useful for computation, because they make dynamical systems easily predictable and attractors can be used as a method of discretizing and storing information.

**Mismatch** Mismatch is a term from VLSI manufacturing and describes the circumstance that no two transistors are exactly equal. Rather each transistor has slightly differing characteristics; the main underlying reason for this is that doping (the addition of electron-rich / electron-depleted material to the silicon) is a statistical process and from the variations in dopant-concentrations (as well as some other factors) arises varying high-level behaviour the units. [CTB07]

If mismatched transistors are used in standard logic circuits, they are always fully 'on' or fully 'off' and the mismatch between them is mostly relevant when it is so severe that a transistor fails to turn on at all. In contrast in analog circuit design, in particular in neuromorphic engineering with sub-threshold transistors [Mea89], the mismatch carries through to have high-level implications: In an array of silicon neurons, each has slightly different parameters, and behaves slightly differently.

While it would be overconfident to argue that this parameter variability is just like the one between biological neurons, it is clear that biological neuronal circuits must also work with units that are not exactly identical (as can e.g. be seen on their varied morphology). It can be hoped that the same organisational principles that allow variable biological neurons to work reliably in conjunction, can allow variable silicon neurons to do the same and that conversely a functioning system of mismatched silicon neurons may shed light on potential functional organisations of variable biological neurons.

**Sampling from a Probability Distribution** Sampling is a method of representing a probability distribution. It is easiest to understand by an example. Say we want to

represent the probability distribution $Q$ over outcomes of a (fair) coin flip. One way of doing this would be to specify the probability of each outcome $x$: $p_{x=\text{heads}} = 0.5$ and $p_{x=\text{tails}} = 0.5$. An alternative way (sampling) of representing this distribution would be to specify a process that for each value of parameter $t$ assigns a variable $y_t$ the value heads with probability one half and the value tails with probability one half. Then a collection $\{y_0,\ldots y_n\}$ is a set of samples from $Q$.

For the fair coin flip it seems unreasonable to represent $Q$ through a process yielding $y_t$, because the detailed description of $Q$ is so compact. For distributions over very large sets however it may well be much more practical to choose the representation by samples. Additionally in many real-world scenarios we can only access distributions through samples (e.g. the distribution over all possible shapes of the character '4').

**Probabilistic Inference** Probabilistic inference is the basic operation in probabilistic reasoning, i.e. information processing under uncertainty. Colloquially one could say that performing inference means to asses the probability of a datum $x$ based on other data $y$ and a model of how datum $x$ relates to data $y$. Notably logical inference is subsumed as a special case under probabilistic inference.

**Perception as Inference** The activity of gaining persistent and consistent perceptions from noisy sensory input, has long been regarded as a kind of 'subconscious inference' [HS25]. An interesting phenomenon in this context is 'perceptual multi-stablity' (see below), as it seems to suggest that the inference is carried out by sampling (if we assume that the inference operates on sets of microstates, that can be categorized into the macro states that on which the multi-stability occurs).

**Perceptual Multi-Stability** Perceptual multi-stability is a phenomenon that arises in psychophysics when a subject tries to reconcile 'contradictory' sensory inputs (i.e. sensory inputs that are inconsistent with the subjects internal model of consistent sensory inputs). Subjects do not report perceiving multiple *contradictory* interpretations of sensory input *at once*, but rather multiple *consistent* interpretations *in series* [Wal78]. E.g. when an object occludes another to one eye only, generally one can see either the occluding object or the one behind it, but not a mixture of both at once.

**Kullback-Leibler Divergence** The Kullback-Leibler (KL) divergence is a non-symmetric measure of difference between two probability distributions. A small KL divergence between two distributions means that they are similar in some sense. In the context of optimal coding, the KL divergence has a somewhat intuitive interpretation. Then the KL divergence is the expected number of bits per bit of message that must be additionally communicated if a message drawn from distribution $q$ is communicated in a code optimal

for a different distribution $p$. Mathematically it has the form

$$KL(q,p) = -\sum_{x\in\Gamma} q(x)\log(p(x)) + \sum_{x\in\Gamma} q(x)\log(q(x)) \tag{1.1}$$

where $\Gamma$ is the support of $q$ and $p$. [Bis06]

**Markov-Chain Monte-Carlo Sampling** A Markov chain [Tie94], is a discrete time system whose current state $x_t$ is only dependent on the directly preceding state $x_{t-1}$ and independent of $t$ or any earlier states $x_{t-s}\forall s > 1$.

Monte Carlo sampling is a method to draw samples from a desired probability distribution. The key principle is the use of a pseudo random number generator (PRNG) with a well known distribution (a system that produces samples from a particular distribution, not necessarily equal to the desired distribution) in combination with a mechanism that 'shapes' the distribution of the samples to have the desired from, e.g. by weighting or rejecting some 'proposed' samples.

A Markov Chain Monte Carlo (MCMC) sampler [MRRTT53] is one particular kind of Monte Carlo sampling method, in which the mechanism for 'shaping' the distribution is based on a probabilistic Markov Chain.

A Markov Chain is defined by its transition probability matrix $T$ over a space of given states $S$. This matrix specifies in its entry $T_{ij}$ for each state $x_i \in S$, how likely it is that the next state will be $x_j \in S$. Formally

$$T_{ij} = p(x_t = x_j | x_{t-1} = x_i). \tag{1.2}$$

$T_{ij}$ naturally induces a limiting probability distribution $\pi_T$ (if the limit exists) over the state space $S$ by the relation

$$\pi_T = \lim_{n\to\infty} T^n p_r, \tag{1.3}$$

[Tie94], where $p_r$ is any probability distribution over $S$. Equivalently one can define $\pi_T$ as the eigenvector of $T$ with eigenvalue one, normalized so as to be a probability distribution ($\sum_x \pi_T(x) = 1$).

To construct an MCMC sampler one first finds a transition matrix $T$ whose limiting distribution $\pi_T$ is the desired distribution $p_d$ (for a given $p_d$ the inducing transition matrix is not unique and there exist many methods of constructing an appropriate $T$ with differing strengths and drawbacks). The MCMC sampler then starts from any state in $S$ and transitions into a next state according to $T$; it is straight-forward to map a uniform PRNG into the state transitions prescribed by $T$. The MCMC sampler then asymptotically produces samples from the distribution $p_d$.

**Markov Random Fields** Markov random fields (MRFs) are a way of representing certain kinds of probability distributions as undirected graphs. In these graphs, conditionally dependent variables share an edge; thus any unconnected variables are conditionally independent given all other variables. (This is called the 'local Markov property' and is the weakest definition of an MRF.) [Bis06]

**Hammersley-Clifford Theorem** The Hammersley-Clifford theorem [HC68] states that the probability density associated with any MRF can be written in the form

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_{c \in \mathsf{cl}} \Phi_c(x_c)\right), \tag{1.4}$$

if the density is strictly positive and where $\mathsf{cl}$ is the set of all cliques in the graph corresponding to the MRF and $Z$ is a normalization constant. In words one could say that the probability distribution over an MRF factorizes into cliques.

**Constraint Satisfaction Problems** A constraint satisfaction problem (CSP) [Mac92] can be defined as follows. Given variables $\vec{x} = (x_1 \dots x_n)$ and constraints $C_1(\vec{x}) \dots C_m(\vec{x})$ find an assignment (or all assignments) of values to $\vec{x}$ so that $C_i(\vec{x}) = 0 \ \forall i$. Because most constraints typically only involve some small subset of the variables, it is intuitive that CSP solvers should be intrinsically parallelizable: Different processes can independently attempt to resolve different conflicted constraints, and occasionally communicate to ensure that shared variables are assigned the same values.

**Stochastic Local Search** Stochastic local search (SLS) [HS04] is an approach to solving CSP problems. An SLS solver starts from a seed proposal solution to the CSP and then successively makes small randomly varying changes to it according to some heuristic (depending on the constraints) in the hope of fulfilling ever more constraints. For some CSP problems SLS solvers employing well chosen heuristics are the current state-of-the-art.

## 1.3 Hardware Platforms for Neural Networks

The computational substrates used to execute neural network (and related) algorithms can be divided into two main groups: Commodity hardware and special purpose 'neuromorphic hardware'.

### 1.3.1 Commodity Hardware for Neural Networks

Currently there are a handful of common hardware architectures that are used for most computations: PCs (von Neumann architecture with up to tens of cores and mostly shared memory), super-computers and sever farms ($10^2 - 10^4$ cores with localized memory) and

GPUs ($10^2 - 10^6$ cores of limited functionality with shared memory). The implicit hope in using neuromorphic systems for some computational tasks is that a system with even more even simpler cores than a GPU and with distributed memory can fulfil certain tasks more efficiently.

### 1.3.2 Neuromorphic Platforms

Here I will review some state-of-the-art neuromorphic systems, summarize their strengths and weaknesses and look at the design philosophies behind them. What makes these systems 'neuromorphic' is that they all provide highly parallel computing platforms without a centralized controller, but with a flat hierarchy of interacting parts. They can be categorized by the following features: analogue vs. digital computation, synchronous vs. asynchronous communication and online vs. offline learning.

**ROLLS** The state-of-the-art online learning system was developed by my colleagues [QMCOSSI15]; it is an analogue asynchronous system. Neurons implement an approximation of an exponential adaptive integrate and fire neuron [BG05] and synapses use a bistable rate- and spike-based learning rule from [BSF07]. The main advantage of these models is their biological realism. The learning rule is also understood to some extent theoretically. The philosophy behind this system is one of exploration: Given a biology-like system, what can we do with it and what do its limitations tell us about biology?

Practically the main limitations of this system however, are its size and inflexibility. There is one neuron model, parameters are largely shared between units and there are relatively few neurons. The inflexibility is inextricably tied to analogue design: Equations are emulated rather than simulated; a circuit implements a particular equation, not another. Analogue computation has the upside of using less energy and potentially providing more resolution per area.

**NeuroGrid and BrainScales** [BGMCCBAIAMB14; Bra] Are also analogue asynchronous systems, but much larger and lack built-in learning functionality. Due to their size they need to address additional problems (especially spike routing) and do not implement plasticity.

**Spinnaker** The opposite end of the spectrum is embodied by the Spinnaker architecture [FGTP14]. Spinnaker is digital synchronous hardware. Commodity ARM processors are interconnected by a special routing fabric, to allow for very large parallel simulations (of among other things neural networks). Spinnaker's main advantage is its flexibility and size. A dedicated spike routing fabric allows fast operation and low energy consumption in comparison to the supercomputers that would typically be used for computations of the scale it is aimed at.

**TrueNorth** TrueNorth [MAAICSAJIGN+14] is the only state-of-the-art neuromorphic system developed privately (by IBM) rather than in academia. It uses digital computation and asynchronous communication and is built with the intention of industrial application. Its main strength is that some machine learning applications have been demonstrated to be compatible with it and it is efficient at running them.

## 1.4 Intuitions Gained in this Project

In this section I give a brief summary of the high-level intuitions I believe to have gained in during this project. These are not exact results, but things that would have helped me do my thesis work more quickly had I known about them from the start.

Two key factors determine a computing architecture's strengths and weaknesses: The computational primitives (the *units* of computation) it offers and the ways in which they are able to interact (the *structure*).

When implementing a particular unit it is useful to have a clear idea of what role it will play in the algorithms one wants to implement. When implementing a particular structure it is good to know what kind of communication it needs to support in application.

On a structural level neuromorphic engineering upholds principles like: Parallelism, efficient communication, fault-tolerance / redundancy instead of high precision. Any subset of these is an interesting topic onto its own.

I believe that massive parallelism and event-based communication are 'co-adapted' features of cortical communication. Parallelism implies locality / non-globality. Being well suited to parallelism means consisting of nearly independent sub-structures and thus requiring only little information exchange between these sub-structures. An algorithm that is well suited to parallel architectures should thus not need to exchange complex messages; therefore event-based communication is intrinsically suitable to parallel architectures.

Event-based communication may also fulfil some algorithmic functions. In unsupervised learning information bottlenecks (like sparsity constraints or 'spiking' (i.e. the Bernoulli experiment after calculating the activation) in ANNs) are important, because they force models to represent data in terms of few underlying primitives allowing them to disentangle the factors explaining the variations in the data [GBB11]; event-based communication may thus be particularly relevant for learning systems. Recently it has been shown that using 'drop out' during test time of a neural network classifier allows the neural network to express confidence estimates on the classification (this is different from the confidence one gets out of a 'soft-max' in the last layer) [GG15]. In some contexts (among them some I will outline in this thesis) the event-based communication forms part of a drop out mechanism for models of biological neurons; perhaps event-based communication (or spiking in the context of neurons) could thus be a necessary step in allowing (biological) neural networks to express confidence estimates.

On the unit level neuromorphic engineering has long been focusing on emulating detailed activation functions of biological neurons. If the goal is computational power (which may not necessarily be the case) this is a great limitation. A more general approach is considering substrate-friendly computational primitives. The flexibility that VLSI gives the designer, should encourage us to think about emulation of computational primitives corresponding to e.g. the activity of groups of neurons rather than just single neurons.

Whatever hardware is used, one does well to consider carefully the computational paradigms that work well for it. E.g. stochastic local search is well suited to highly parallel structures; von Neumann architectures are good for e.g. numerical integration of low dimensional differential equations.

We have a poor understanding of the relevance of time in cortical processing (in contrast we have a clear idea about the organisation in space of computation and have made use of this). In this thesis I reduce the timing of events essentially to their ordering / probability of approximate coincidence, but very likely there are a lot of useful things to be done by actually using time properly.

In computer science (artificial neural networks) and in computational neuroscience (spike response model) there is in general a fashion of reducing timing to probability (also in models of spike-time dependent plasticity (STDP), where the synapses may carry actual timing information, if the neurons are probabilistic). A small step into the direction of timing-aware neural networks are recurrent artificial neural networks – however the training algorithm usually used on them (back-propagation through time) is unnatural (non-causal). Good theories in this area might be fruitful.

## 1.5 Structure of this Thesis

The basic build-up of the different chapters of this thesis is roughly from more theoretical to more applied, with some adjustments to facilitate understandability.

In chapter 2 I briefly summarize recent related work.

In chapter 3 I present an abstract model of neural networks under oscillatory inhibition. I find that it implements an MCMC sampler.

In chapter 4 I study the model's capability to perform inference and investigate how to learn connection structures that induce particular probability distributions (learning from samples).

This suggests efficient 'hardware-friendly' methods of implementing artificial neural network algorithms that I study in chapter 5. I investigate the viability of this approach under physical limitations (neuron/synapse complexity) in simulation.

In chapter 6 I generalize my approach from MCMC to SLS algorithms, a kind of network based CSP solvers. The presented massively parallel algorithms could plausibly surpass current state-of-the-art in terms of speed and power-consumption under reasonable

assumptions about the physical implementation of recent neuromorphic architectures.

In chapter 7 I conclude my thesis and discuss the impact of the work presented.

Finally in the appendices I give an overview of two related topics I worked on during my PhD time: Finite-size mismatched WTA in appendix A and an application of the network model to the putative functional role of gamma oscillations in perceptual multistability in appendix B.

Chapter 2

# Event-Based Massively Parallel Algorithms

## 2.1 Introduction

Several algorithms have been formulated successfully for event-based hardware. I will review the examples I am aware of and explain briefly why these have so far not brought about a wide-spread adoption of neuromorphic platforms. A related field are algorithms that have been formulated for event-based models of biological neurons - this is a wide field and I will name the ones that seem most relevant in the context of my work.

### Algorithms with Implementation on Event-based Hardware

**Finite State Machines** On the example of finite state machines [Nef10] demonstrated that analogue spiking neurons can approximate linear threshold units well enough to implement a controlled complex dynamical system. Practically finite state machines are routinely implemented with much greater efficiency in traditional ways (one should add that the creation of an efficient FSM was not the aim of [Nef10]).

**Hopfield Nets** Another use of large groups of spiking neurons behaving close to mean-field is to implement attractor dynamics and a kind of 'content-addressable memory' (CAM) [GCMDBG12] in the style of hopfield nets [Hop82]. This kind of CAM has the issue that it can store only non-overlapping (or slightly overlapping) patterns and the use of mean-field trades off the efficiency of event-based communication for conceptual simplicity.

**Perceptron** [Ste13] used an event-based implementation [BSF07] of the perceptron algorithm [ABR64] in conjunction with high-dimensional random projections (for linear separability) [MBWWDMF13] to classify MNIST digits. I am not aware of a demonstration of this type of classifier in which it performs near state-of-the-art level (however the architecture can be scaled up indefinitely, which may yield better performance) and generalization

to multiple layers is non-trivial (though multiple layers may not be necessary given the alternative mechanism for separation of not linearly separable classes).

**Artificial Neural Networks** Some *synchronous* artificial neural network algorithms going back to Boltzmann machines [HS83] could be classified as event-based algorithms, because the output of the interconnected units ('neurons') they are built up from is discretized to two levels (output and no output / event or no event); in other words the communication between units could be easily handled by an Adress-Event Representation (AER) router. However in their standard implementations many units are updated simultaneously; this simultaneity is unimportant, also in training [Hin02; NDPKDC14]. [ONLDP13] made use of the fact that the synchronous updates are unimportant in RBMs and formulated an event-based asynchronous RBM for a sensor-fusion and classification experiment.

Several very recent papers have used similar *asynchronous* event-based approaches to implement feed-forward neural networks for classification [DNBCLP; GBVRGPGD14]. Finally [SNPGFL15] uses a similar formalism to [ONLDP13] to implement a DBN on Spinnaker hardware (a hybrid synchronous / asynchronous platform). I believe that this is a very promising approach; my main concern in this context is the use of high-resolution random numbers for each neuron update and the persisting use of a synchronous domain that cannot exploit activation sparsity. The deterministic network of [DNBCLP] however overcomes the random-number problem as well (while [DNBCLP] only studies classification, generalization to auto-encoding is straight-forward) – (pseudo-)probabilistic behaviour may be desirable in some contexts though, e.g. in the form of drop out, to obtain confidence estimates [GG15].

## Other Event-based Algorithms

**SRM-based Algorithms** The group of Wolfgang Maass has in recent years produced a host of algorithms implementable on a particular model of spiking neurons, the 'spike response' model (SRM)[JLG04]. These algorithms include MCMC sampling [BBNM11], expectation maximisation [NPM09] and an MCMC-based constraint-satisfaction problem (CSP) solver [PBM11]. The SRM model is an intrinsically probabilistic neuron model; i.e. it is not a full physical, but a phenomenological model of neurons. As such this model and the algorithms developed for it do not address physical implementability; the – in engineering terms – key problem of generating good independent probabilistic behaviour in a large population of neurons is not dealt with in this approach.

## 2.2 Differentiation from and Relationship to Prior Work

In the preceding section I briefly described several event-based algorithms (mostly originating from a neuroscientific background) that have to date not been implemented with the hoped-for efficiency gains in hardware. Notably I develop some of the same algorithms in event-based formulations in a different theoretical framework (namely an MCMC sampler as [BBNM11], various ANNs as [ONLDP13] and a CSP solver as [PBM11]). This immediately poses the question why another theoretical approach to the same algorithms should be called for.

A first answer applies primarily to the third chapter of this thesis: The differing theoretical framework is required to explain a different physiological phenomena. For example I examine the physiological phenomenon of oscillatory inhibition in an attractor network; the justification of considering event-based MCMC samplers in this context is that an MCMC sampler is a good approximation to the behaviour of this physiological model. (In contrast [BBNM11] primarily conveys the point that MCMC sampling in spiking neural network is in principle possible and posits that noisy neuronal dynamics could be the underlying physiological process that support the algorithm.)

Partially this answer also applies to later chapters. Given that the physiological process of oscillatory inhibition seems to support MCMC sampling, an obvious follow-up question is whether it supports very closely related algorithms such as ANNs and CSP solving.

A more fundamental argument is suitability for hardware implementation: How readily can the algorithm / the computational primitives it draws on be built as a physical system? E.g. the ease of implementing exponential I-V transfer curves in VLSI (in the form of subthreshold transistors) is likely one of the original inspirations for neuromorphic engineering [Mea89].

The specific computational primitive whose implementation poses a particularly great challenge in [BBNM11; PBM11; ONLDP13; NDPKDC14; SNPGFL15] is the noise generator or pseudo-random number generator. The quality ('randomness' of the rng) is *not* the central concern here (in fact in quasi-Monte Carlo methods [Caf98] low-discrepancy sequences [Hal60; Sob67] are used as underlying noise rather than pseudo-random numbers to decrease the variance of the sampler). The central problem is how to effectively integrate the noise sources into the system and build them as cheaply as possible. In the framework of this thesis the very cheap noise source of mismatched oscillators is shown in simulation to work to a good approximation of the theoretical model.

One class of algorithms I mentioned (what I dubbed synchronous ANNs) such as [HS83] have found highly effective implementation e.g. on GPUs. These implementations however do not make use of the fact that the underlying algorithms are often sparse (most of the time most neurons are inactive; connectivity may also be sparse). Due to this sparsity there is a lot of room for improvement in terms of energy consumption of the supporting

hardware: Ideally computation should only be done when a neuron is active. This can probably only be done in event-based hardware.

Neural network based CSP solvers have been previously suggested [PBM11; HT85]. In contrast to these I start from a sequential stochastic local search CSP solver (a state-of-the-art SAT solver) and cast it into the form of an event-based network. The resulting network has the distinguishing features that fully satisfying solutions are stable and that it can reach state-of-the-art performance running on hardware clocked at around 1 MHz.

Chapter 3

# Formal Setting: The Poisson-Bernoulli-Maximum Net

Sections 3.3 - 3.5, 3.7 and 3.9 are based on the paper "Rhythmic inhibition allows neural networks to search for maximally consistent states" by Hesham Mostafa, Lorenz K. Müller and Giacomo Indiveri [MMI15b]; section 3.8 is based on the paper "Recurrent Networks of coupled WTA-oscillators for solving constraint satisfaction problems" by Hesham Mostafa, Lorenz K. Müller and Giacomo Indiveri [MMI13].

## 3.1 Introduction

How to enable a system that exhibits multiple attractor states to transition between them, is a complicated question. An even more daunting version of it is the following: Given a collection of $N_W$ WTA, each having $N_s$ stable states, how can they be connected into a network that explores the full $N_s^{N_W}$ possible system states in a meaningful way? In short, one possible method to achieve this is to periodically destabilize the attractor state of each WTA, forcing it to periodically re-evaluate its inputs and to briefly pause its output [MMI13].

In this chapter I will propose a theoretical framework in which to study this mechanism. First I introduce the notion of continuous time sampling. By moving from periodically active components to components with Poisson-distributed activity periods and by reducing the dynamical features of the attractor (WTA) network to instantaneous jumps to its limiting states, I show that such a 'PBMnet' approximately implements an MCMC sampler. In simulation I verify that this approximated network is behaviourally a good fit to the periodic system it models. Further I provide some intuition on what the probability distribution sampled by the MCMC sampler looks like and note that the PBMnet can be regarded as a model of WTA circuits under oscillatory inhibition as well as a model of

spiking neural networks.

## 3.2 Continuous Time Sampling

In this section I introduce the concept of continuous time sampling (CTS) (that I have not seen defined anywhere else). While it is very intuitive what one would mean by CTS, I hope to avoid possible confusion in the following by giving a definition first.

When trying to assess the underlying probability distribution of a discrete sampling process one can accumulate samples in a histogram over the entire space of permissible states and then normalize this histogram after having observed sufficiently many samples. An analogous process for the time-continuous sampler integrates the time the system spends in each state over a sufficiently long time and then normalizes this histogram.

There are two ways to interpret this time-continuous sampling process: Each time the system switches state constitutes a weighted sample of the state that is left (consider also ending the sampling process as switching the state), where the weight is given by the time spent in that state. Alternatively one could say that the dynamical system proposes a sample at every infinitesimal time point and the sum performed to accumulate the histogram, becomes an integral over time.

Either way, I define the normalized histogram of times spent in a certain state as a probability distribution over these states and I interpret the unfolding time course of the dynamical system as a set of samples that induce this distribution.

Formally these interpretations could be written as follows:

$$p(y) = \lim_{T_s \to \infty} \frac{\int_0^{T_s} \delta(x(t) = y) dt}{T_s} = \lim_{T_s \to \infty} \frac{\sum_{x_i} \delta(x_i = y) t_{x_i}}{T_s} \tag{3.1}$$

where $p(y)$ is the probability of state $y$ under the dynamical system, $x(t)$ is the time course of the state of the dynamical system and $\delta(q)$ is one if $q$ is true and zero otherwise. The $x_i$ are all members of the set of states that were occupied by the system just before a state transition and the $t_{x_i}$ are the corresponding times since the last transition. Practically $T_s$ is a large but finite sampling time.

A justification why this definition is 'sensible' could be given as follows. For a *discrete* time sampling process the probability $p(y)$ of sample $y$, expresses how likely it is to find the system in state $y$ if it is probed at a random time step $t$ of its dynamics. Using the above definition of $p(y)$ for a *continuous* sampling process, $p(y)$ expresses how likely it is to find the system in state $y$ if it is probed at a random time point $t$ of its dynamics.

An alternative approach is to eliminate the meaning of 'time' in the sampling process and go to an event-based discrete time sampler that represents the distribution [BBNM11]

$$p(y) = \lim_{T_s \to \infty} \frac{\sum_{x_i} \delta(x_i = y)}{Z}, \tag{3.2}$$

where $Z$ is an appropriately chosen normalization constant.

In addition to having the property of generating a specific limiting distribution over long sampling times, a useful *pseudo-random* sampler should provide samples, so that samples are independent over sufficiently many time steps and should consequently provide unbiased samples from the probability distribution after some burn-in time (in which the initial state is forgotten). Similarly a continuous time sampler should provide samples that are independent if a sufficiently long time passes between them.

## 3.3 The Poisson-Bernoulli-Maximum Network

In this section I will describe a network, the poisson-bernoulli-maximum network (or PBM-net), distantly inspired by biological neural networks. The connection to biological systems will be explained in sections 3.5 and 3.10, but first I will define the abstract system here and explore some of its properties.

A PBMnet, see figure 3.1, consists of interconnected nodes that represent populations of neurons (or single neurons). In the following superscripts will be used to denote the membership of a quantity to a node. Each node represents a variable. The state of node $i$ at any time is given by two vectors $\vec{v}^i(t)$ and $\vec{I}^i(t)$: $\vec{v}^i(t)$ represents the discrete output value of the variable corresponding to the node in a one-hot encoding (one entry is one, the others zero). If the $k$th entry of $\vec{v}^i(t)$, i.e. $v^i_k(t)$ is one, the corresponding variable has state $k$. The input vector $\vec{I}^i(t)$ keeps track of inputs to the node. The vector $\vec{I}^i(t)$ has one entry $I^i_k$ for each possible value $v^i_k$.

Intuitively one may think of the vector $\vec{v}$ as the vector that identifies in a one-hot encoding which of $k$ competing populations receives the highest input in the input vector $\vec{I}$ (with some complications that will be explained). This (and only this) 'winning' population then may be able to influence the competitions in other nodes.

The nodes communicate through connections between different possible values; each connection is weighted according to a weight tensor $W_{i,j,k(t),l(t)}$ that specifies the impact state $v^j_l$ of node $l$ has on state $v^i_k$ of node $i$.

A PBM node can only change its output value at special time *points* that I will refer to as 'spike windows' (because in the single-neuron interpretation of a PBM node, they correspond to times at which the neuron may spike) or 'update times'. Let $s^i(t)$ be the function that takes value one, in spike windows of $i$ and zero otherwise. The spike-windows $S^i = \{t|s^i(t) = 1\}$ of node $i$ are Poisson-distributed in time with a fixed average density of $R$ (the spike-rate of the node); notably these spike-windows are determined 'a priori' in the sense that they are independent of the input node $i$ receives. In contrast to these windows of opportunity for state changes, the times at which changes into a particular state occur, are *not* Poisson distributed in general.

The 'a priori' determined spike windows are a key difference to models in the vein

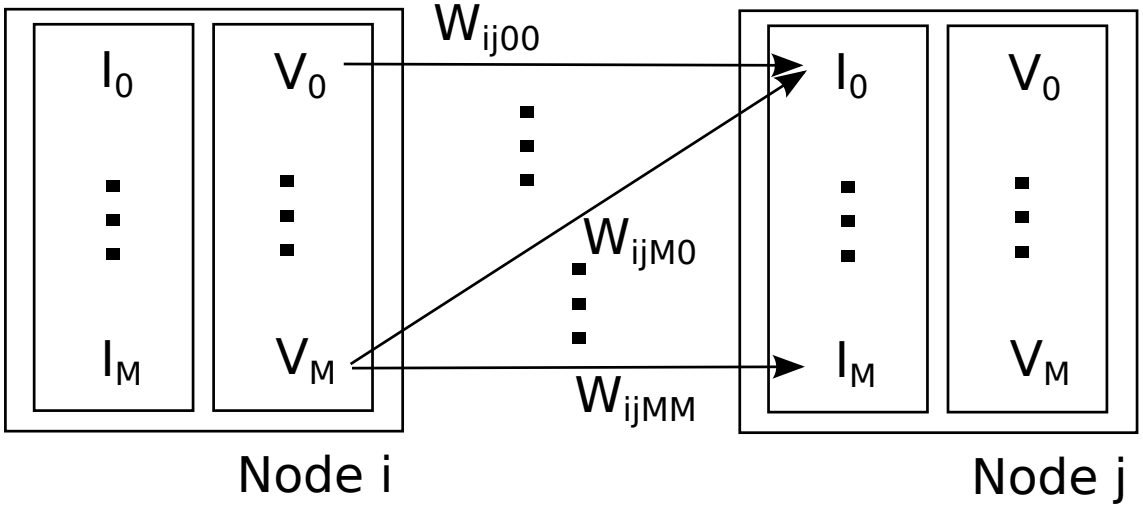



**Fig. 3.1:** *A PBMnet consisting of two nodes. In $I_k$ inputs from 'active' presynaptic values are aggregated; at 'spike-windows' the node 'activates' the value with the highest input.*

of [BBNM11] or [NPM09]; these models introduced a particular interpretation of spike sequences as samples, but the mapping from a spike sequence to a sequence of samples differs from the one used here.

At a spike window node $i$ takes a new value $v_x^i(t)$ depending on its recent inputs. Specifically it takes the value $v_x^i(t)$ for which node $i$ received the highest weighted input, $x = \text{argmax}_k(I_k^i(t))$; the input $I_k^i(t)$ to node $i$ is given by

$$I_k^i(t) = \sum_{j,l} v_l^j(t) W_{i,j,k(t),l(t)} b_p \tag{3.3}$$

Where the sum over $j$ goes over all upstream connected neurons and $l$ over all their possible values. $b_p$ is a Bernoulli variable (of value zero or one) with probability $p$. In words, at a spike window, each possible value of the node sums its current input from connected variable values, weighted by the weight tensor, with a probability $p$ to ignore some inputs; then the value with the highest input becomes the node's new value.

Note that it is irrelevant how often an upstream neuron has changed its state ('spiked') since the downstream neuron's last spike window, only the current state at the update time matters. If there is ambiguity about which value received the highest input (i.e. if there is no unique $\text{argmax}_k(I_k^i(t))$ , the node maintains its previous value (though 'tie-breaking' mechanisms other than this 'hysteretic' one could also be used).

Since the spike-times are independent of other state variables, we can analyse this continuous time system as a discrete time system: Each discrete time-step corresponds to a time $t_s$ at which one of the nodes may change its state. Then we can say for the discrete time system that at each time-step one (uniformly) random node undergoes a state change

(this state 'change' may be the trivial transition from a state to itself).

Any requirement we would like to impose on the mutual information between samples at a certain time distance can now be addressed in terms of the discrete time system and then translated into probabilistic statements about the continuous time system: For a fixed number of discrete time steps there is a probability distribution over how much time it takes in the continuous time system for these steps to occur.

Notably the corresponding discrete-time system is Markovian, which simplifies its analysis. - I will show later on that the Poisson distribution of the spike-windows which is the core justification for the Markovian analysis, does not appear to be critical however, in practice: Even spike-window distributions with high temporal correlations give rise to nearly unchanged behaviour on the system level. This indicates that the connectivity of a PBMnet is the key to shaping the probability distribution over different system states.

## 3.4 Modelling a PBMnet as an MCMC Sampler

In section 3.2 I outlined a method to define or measure the probability distribution $p_{CTS}$ produced by any CTS. While this method makes sense as a definition, it is of relatively little practical use: To find out what probability distribution is encoded by e.g. a PBMnet with a given connectivity structure, this method requires us to simulate its continuous time behaviour.

Here I will describe a simpler method of obtaining the limiting distribution of a PBMnet. The key question is how to construct the transition matrix $T$ of the corresponding MCMC sampler, since this fully determines its behaviour. Let the system be in state $s_i$, what is the probability that it will transition into state $s_j$? We study three cases for different relationships between $s_i$ and $s_j$.

1. *$s_i$ and $s_j$ differ in more than one variable.* In this case the probability $T_{ij}$ is set to zero, because it is impossible for multiple variables to change their state at the same time in a PBMnet (time is continuous and state changes are instantaneous, so that the probability of two occurring simultaneously vanishes).

2. *$s_i$ and $s_j$ differ in exactly one variable.* In this case $T_{ij} = P(v_{up}) \cdot P(v_{new} = v(s_j)|s_i,v_{up})$, where $P(v_{up})$ is the probability that the variable $v$ in which $s_i$ and $s_j$ differ is the one that changes its state and $P(v_{new} = v(s_j)|s_i,v_{up})$ is the probability that $v$ takes the value it has in $s_j$ given that the system state at the change is $s_i$ and that $v_{up}$ is the variable that changes its state. I will shortly address how to construct these probabilities from the network connectivity.

3. *$s_i$ and $s_j$ are the same state.* In this case we can construct $T_{ij}$ using the fact that the previous item implicitly defines the probability $p_{\neq} = P(s_i \neq s_j)$ by the relation $p_{\neq} = \sum_{j \neq i} T_{ij}$, so that $T_{ii} = 1 - p_{\neq}$ by virtue of the normalization of $\sum_j T_{ij}$.

The probability $P(v_{up})$ is simply set to $\frac{1}{n_N}$ where $n_N$ is the number of nodes in the PBMnet: Each node updates at Poisson distributed time points and it is therefore equally likely that any node is the next one to update.

I will consider the case where each variable has two possible states and all weights are of equal absolute value for notational simplicity. The general case without these restrictions is straight forward to write down based on this simple case, but is unnecessarily difficult to read. Instead of summing up weights we simply count numbers of inputs to a particular state. Let $n$ be the updating node and let $u(n)$ be the number of potential inputs to state 0 of $n$ and $t(n)$ the total number of potential inputs to $n$. The number of potential inputs to state 1 is then $t(n) - u(n)$. In formulas I define

$$t(n) = \sum_{j,k,l} v_l^j(t) W_{n,j,k,l} b_p, \tag{3.4}$$

$$u(n) = \sum_{j,l} v_l^j(t) W_{n,j,0,l} b_p. \tag{3.5}$$

As defined in equation 3.3 every potential input is set to zero with probability $p$. Therefore the node receives an effective number $t_{\text{eff}}(n)$ of inputs. This number is distributed according to

$$P(t_{\text{eff}} = k) = \binom{t(n)}{k} (1-p)^k p^{t(n)-k}. \tag{3.6}$$

The aforementioned conditional transition probability $P(v_{new} = 0 | s_{\text{prev}}, v_{up})$ is simply the probability that most of the received inputs go to state 0:

$$P(v_{new} = 0 | s_{\text{prev}}, v_{up}) = \sum_{k=0}^{t(n)} \underbrace{\binom{t(n)}{k} (1-p)^k p^{t(n)-k}}_{P(t_{\text{eff}}=k)} \underbrace{\sum_{l=0}^{k/2-1} \frac{\binom{u(n)}{k-l}\binom{t(n)-u(n)}{l}}{\binom{t(n)}{k}}}_{P(u(n) > t(n) - u(n))}, \tag{3.7}$$

which simplifies to

$$P(v_{new} = 0 | s_{\text{prev}}, v_{up}) = \sum_{k=0}^{t(n)} (1-p)^k p^{t(n)-k} \sum_{l=0}^{k/2-1} \binom{u(n)}{k-l} \binom{t(n)-u(n)}{l} \tag{3.8}$$

and finally yields

$$T_{ij} = \frac{1}{n_N} \sum_{k=0}^{t(n)} (1-p)^k p^{t(n)-k} \sum_{l=0}^{k/2-1} \binom{u(n)}{k-l} \binom{t(n)-u(n)}{l}. \tag{3.9}$$

Note that the above is dependent on the previous state and the target state because $u(n)$ (and $t(n)$) depends on them (as can be seen in equations 3.4 and 3.5). The fact that the inner sum is taken up to $k/2 - 1$ rather than $k/2$ represents the tie-breaking mechanism

that was used.

Using the above construction, we can evaluate the probability distribution induced by a certain connectivity $W$ by finding the limiting distribution of the associated MCMC sampler. To get this one still has to solve a large system of linear equations (the limiting distribution is the eigen-vector with eigen-value one of the transition matrix).

This is somewhat dissatisfying as the problem of how to construct a weight matrix that produces a desired distribution (or models a set of given samples) is still open (because it is difficult to invert the process that lead from connectivity to limiting distribution). In the next chapter I will tackle this problem from a different avenue and show that contrastive divergence learning happens to be an effective method of training PBMnet.

## 3.5 An Oscillator Driven WTA

In the following I describe an oscillator driven WTA (oWTA) network that can be modelled well by the PBM network and originally I developed the PBM network as a tractable variant of oWTA networks. – While it is easy to have an intuition about what states a given oWTA network favours, it is difficult to asses what probability it will assign to each. I will show that the MCMC sampler constructed from the PBM network matches the empirical distribution of the oWTA network closely. This will in the next chapter allow me to train oWTA networks by samples from a desired distribution efficiently. The oWTA network was conceived for [MMI15b] as a model of gamma oscillations by Hesham Mostafa and the remainder of this section and sections 3.6 and 3.7 thus primarily describe his work.

The generic oWTA looks mostly like a normal WTA: There are multiple excitatory populations that recurrently excite themselves and recurrently inhibit each other by way of a shared inhibitory population. The connection weights are set so that the WTA has stable attractors where one of the excitatory populations is active alone. In addition to this, the oWTA has some mechanism that destabilizes each these attractors after a characteristic time $T$ (I will refer to $1/T$ as the frequency of the oWTA). One way of implementing this is to have all populations of the WTA receive a periodic input, see Fig. 3.2.

On an abstract level, we could say that the oWTA is at any given time representing the value of a variable, where the value is given by the identity of the excitatory population that was most active before the current oscillation phase started.

In greater detial, the behaviour of an oWTA is the following: At the beginning of one oscillation phase the current attractor is destabilized. The destabilizing mechanism is released and the system starts moving towards one of its stable states, namely the one in which only the excitatory population that gets the strongest input is active. The phase ends as the destabilizing mechanism begins to operate once more.

Dynamical systems were modelled by Euler integration of interconnected linear threshold units, as described in [MMI15b] unless specified differently.

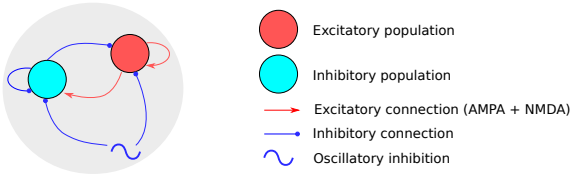



**Fig. 3.2:** *oWTA node*

## 3.6 Coupled oWTA[1]

For useful behaviour to emerge, multiple oWTA nodes need to be connected. By creating such connections the values that different variables (embodied by different oWTA) take start to influence each other, see Fig. 3.3.

Varying phase-relations between connected oWTA lead to a constantly varying effective connectivity that allows the network to search for maximally consistent states with respect to the constraints embodied by inter-oWTA connnections. In later sections I will use the word constraint to refer to inter-oWTA connectivity as it is shown here.

## 3.7 oWTA as a Functional Model of Gamma Oscillations[2]

In [MMI15b] we argue that oWTA networks model the salient features of the rhythmic inhibition that underlies Gamma oscillations; the following are biological justifications for the various modeling assumptions:

1. *WTA circuits are local neural circuit motifs*: WTA circuits are potential cortical circuit motifs [DM04]. The WTA circuits can be replaced by any neural circuit as long as this circuit displays a number of distinct firing patterns based on external input and a memory of past firing patterns. Some distinct part of each firing pattern should be characterized by a high enough firing rate so that it can influence the firing pattern in another neural circuit when the two are coupled.

2. *Oscillatory inhibition is local to each WTA*: Gamma oscillations typically have a local origin [BW12; SS00; ALCRTW11]. Gamma oscillations in the local field potential typically arise from the rhythmic firing of basket cells that have predominantly local arborizations [Fri09]. It is a general phenomenon that faster rhythms tend to develop locally [BD04].

3. *Local oscillatory inhibition is strong enough to shut down the activity in the local circuit*: There is strong evidence for the involvement of interneurons containing the calcium binding protein Parvalbumin in the oscillatory discharge underlying Gamma oscillations [SZYD09; CCMKZDTM09]. Interneurons expressing Parvalbumin, such

[1] this section describes the work of Hesham Mostafa
[2] this section describes the work of Hesham Mostafa

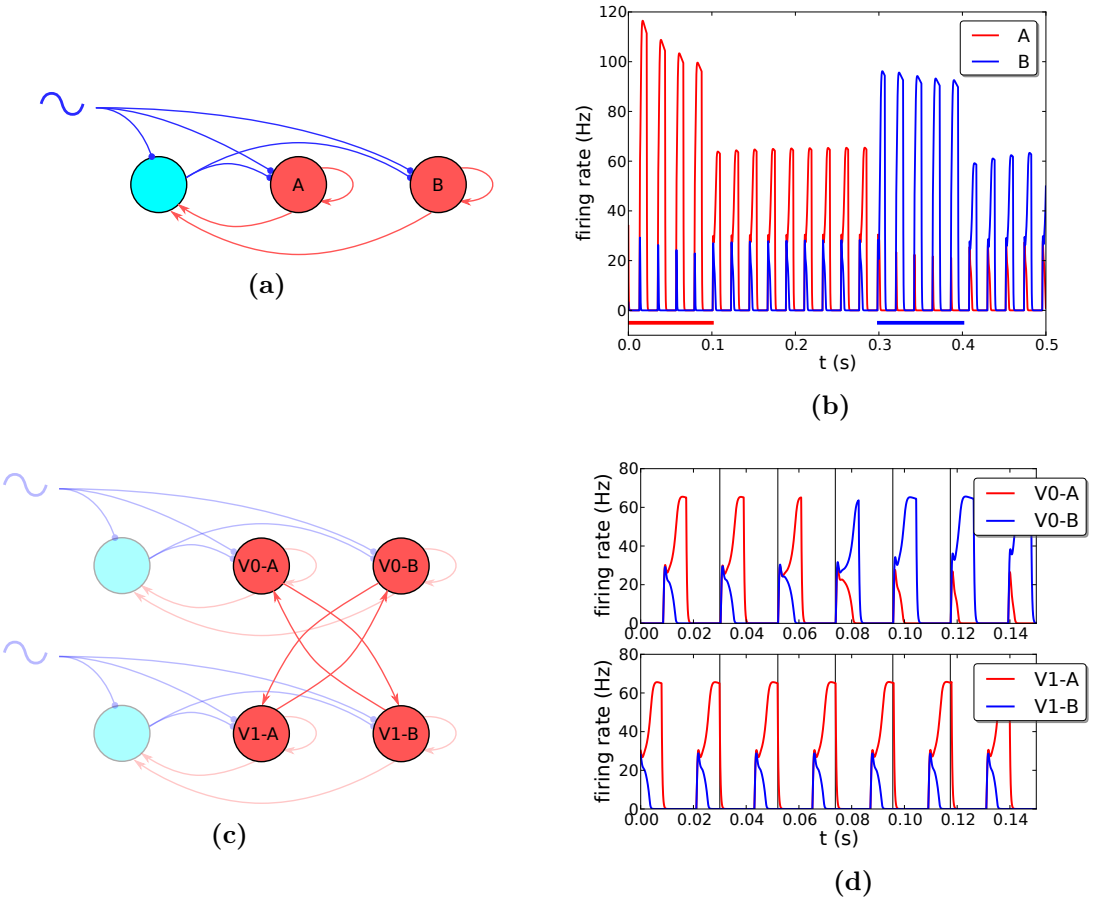



**Fig. 3.3:** *(a) A two-node WTA circuit, modulated by oscillatory inhibition, that represents a binary random variable. (b) Simulation results of the two-node WTA circuit. Oscillatory inhibition periodically shuts down all activity. When released from inhibition, the WTA selects a winner. External input (colored horizontal bars) can bias the winner selection process. In the absence of external input, the identity of the winning excitatory population is maintained across the inhibition cycles. (c) Coupling two WTA circuits, that represent two variables V0 and V1, to enforce the consistency condition V0≠V1. (d) Simulation results of the network in (c). Initially, the consistency condition is violated (i.e., V0=V1). As the frequency of the oscillatory inhibition is slightly different in the two WTA circuits, the phase difference between the inhibitory cycles in the two WTA circuits gradually changes. Eventually one WTA circuit is able to switch the state of the other WTA to satisfy the consistency condition. Vertical bars are visual guides to highlight the end of the high phase of oscillatory inhibition in the V0 WTA circuit.*

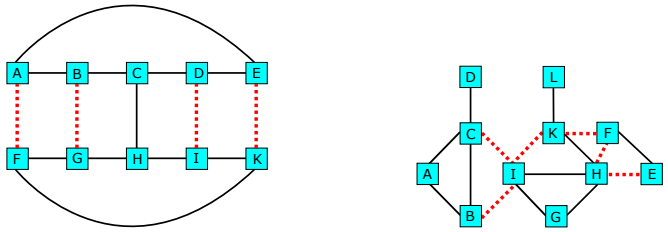



**Fig. 3.4:** *Ising model type problems. Each square indicates a binary variable like in Fig. 3.3a; solid black lines denote a constraint requiring two variables to be equal, dashed red lines a constraint that requires two variables to be unequal. In both problems, all states violate at least one constraint.*

as basket cells and chandelier cells, mainly target the soma, the axon initial segment, and the proximal dendrites of the excitatory principal cells [MTRWGSW04] making them particularly effective in rhythmically silencing their target neurons.

4. *The local oscillatory inhibition waveforms have different frequencies*: There is evidence that Gamma oscillations recorded from even nearby regions in the same cortical area have significantly different frequencies [RM10]. Phase relations between Gamma oscillations recorded from nearby points in the cortex are continuously changing [WSOSDEF07; Fri09], and the phases of local cortical Gamma rhythms vary in an irregular manner [BSXSS10]. Different oscillation frequencies is one simple way to obtain continuously changing phase relations. The different local rhythms can be highly coherent and we investigate the effect of this coherence on the model behavior. The only requirement of our model is that the space of possible phase relations is continuously explored with no perfect and persistent phase lock between any of the local rhythms.

## 3.8 State Probability and the Number of Violated Constraints

In [MMI13] we studied the relationship between the probability of an oWTA network's state and the number of constraints it violates. Let $E(s)$ be a function that maps a network state $s$ to the number of constraints it violates (I will refer to $E$ as the energy of a state, although it is not strictly speaking an energy for the oWTA network). For the problem in Fig. 3.4a, we observe that the average time the network spends in states with $E(s) = E$ is $t(E) \approx c_1 \exp(-c_2 E)$ as can be seen in Fig. 3.5a.

States that violate more constraints are visited for a shorter time, because the higher the number of violated constraints, the more rapidly the variable values change as there are more possible phase relations that can emphasize a violated constraint.

The network spends almost equal times in complementary states that violate few constraints. Complementary states are maximally different but the network is able to traverse

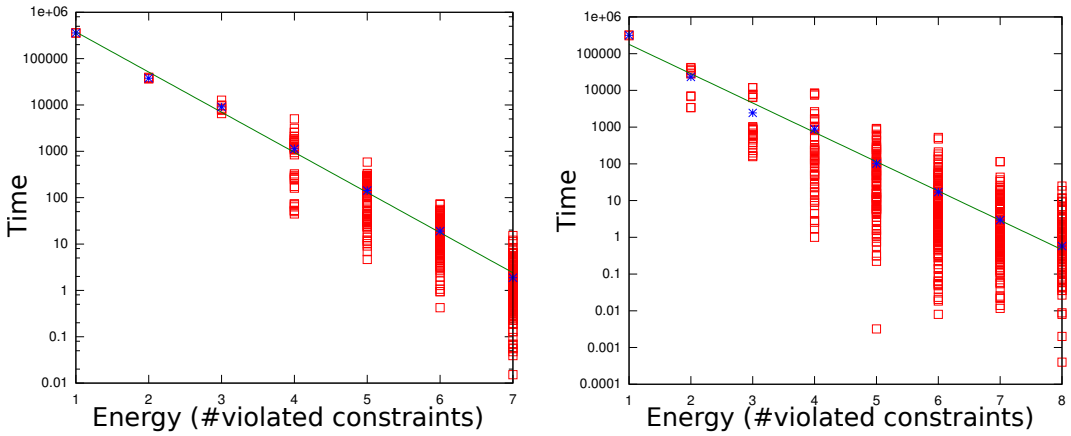



**Fig. 3.5:** *Behavior of two oWTA networks representing the CSPs in Fig. 3.4. Red squares are data points (the time the network spent in one particular state), a blue star is the average time spent in states of equal energy and the green line is an exponential fit to the blue stars. (a) Note that at energies 1 and 2 there are two complementary states* each *that are visited almost equally often. (b) Not all assignments of 'energy' 2 are equally probable in this case (not an artefact of finite samples, but systematic) as can be seen in the bimodal distribution there.*

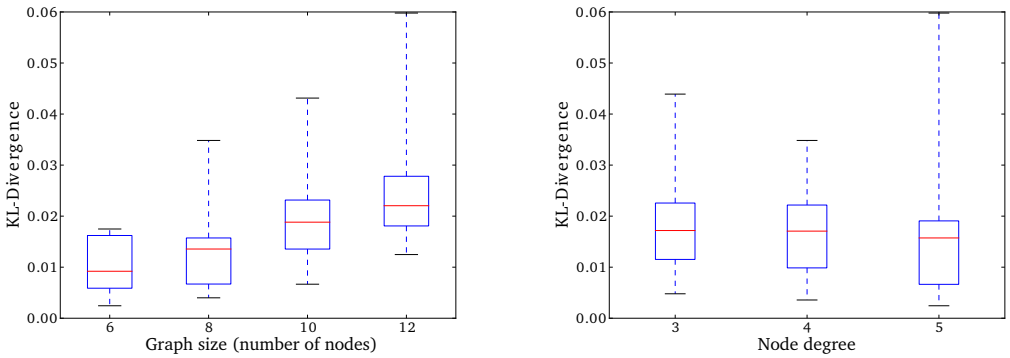
the space of intervening states (which may violate more constraints) in order to visit the complementary states almost equally often. – The reason for this nearly exponential relationship will be explored in the next chapter.

Note that in [MMI13] the oscillators driving the different oWTA nodes were weakly coupled.

## 3.9 oWTA and PBMnet

The oWTA network differs from the PBMnet in two ways. The update (or spike) times occur at a fixed period in the oWTA network, rather than being poisson-distributed as in the PBMnet. Secondly the instantaneous max-function with arbitrary memory of the PBM node is replaced in the oWTA node by several mechanisms that act together to produce a similar effect: Each oWTA node is state-holding and keeps communicating for a long time after it has assumed a new state (a longer memory is unnecessary because nodes update at approximately equal frequencies); after the attractor state of an oWTA node has been destroyed by the oscillatory inhibition a dynamical system computing a max-integral over some time window selects the new state of the node.

A theoretical analysis of the discrepancy between the PBMnet and the oWTA network seems infeasible to me; however, a comparison can easily be done in simulation. This is an interesting comparison to make, because if they are sufficiently similar the PBM formalism yields a quickly computed approximation to the distribution that a given oWTA network




**(a)** *Grouped by number of nodes in graph* **(b)** *Grouped by node degree*

**Fig. 3.6:** *Comparison of distributions produced by oWTA and PBMnet using the KL divergence* $KL(p_{oWTA}||p_{PBM})$.

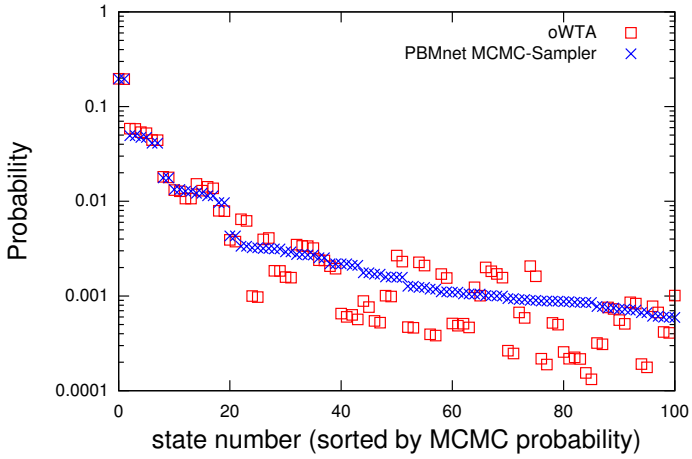


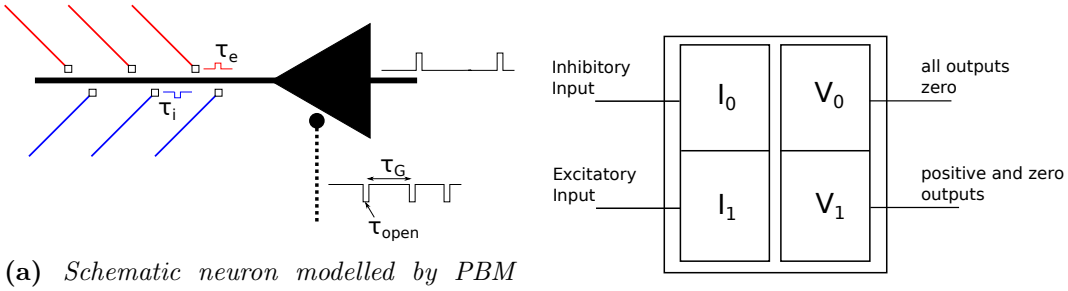

**Fig. 3.7:** *Direct comparison of the PBMnet and measured oWTA distributions for an example random graph with ten nodes, for the 100 most likely states. States are ordered by probability under the PBMnet MCMC sampler; lower probability states (clipped) have smaller absolute error.*

samples from (which can only be accessed by simulating the oWTA network otherwise).

To evaluate how well the stable distribution of the MCMC sampler induced by a PBMnet matches the empirical distribution produced by an oWTA network, I provide plots of Kullback-Leibler (KL) divergences between the limiting distribution of the PBMnet and the oWTA network for randomly generated graphs[3] (fig. 3.6) and the measured normalized histograms for the graph in fig. 3.5a in fig. 3.7).

Clearly the PBMnet is a useful model of the oWTA network in the sense that it provides an easily computable approximation to the distribution from which the oWTA network samples. This is not very surprising: Although the PRNG induced by the incommensurable oscillators at first look would seem to be of very poor quality, apparently its temporal

[3]random graphs generated by Hesham Mostafa




**(a)** *Schematic neuron modelled by PBM node*

**(b)** *The corresponding PBM node.*

**Fig. 3.8:** *(a) A neuron under gating inhibition that releases at poisson distributed times ($\tau_G$, with very short times disallowed) for a short time ($\tau_{open}$). At these times it evaluates whether the sum of EPSPs (time constant $\tau_e$) or IPSPs (time constant $\tau_i$) is greater. Typically assume $\tau_i \approx \tau_e$, $\tau_{open} \ll \tau_e$ and $\tau_e < \tau_G$. (b) If the neuron gets above threshold input (i.e. more input at terminal $I_1$) it emits a spike (i.e. goes to state $v_1$). The limited probability of that spike having an impact in a network of intermittently inhibited neurons is modelled by the Bernoulli variable that randomly sets spike weights to zero.*

correlations are not critical; this makes sense in light of the fact that an MCMC sampler always (independently of its PRNG) has a high correlation between subsequent samples.

## 3.10 Single Neurons as PBM Nodes

The PBM model is not only relevant for oWTA: It is applicable to any system that can be approximated as a thresholding max-function combined with unreliable message transmission. In particular the thresholding max-function a neuron applies when it decides whether or not to fire could serve as such. In figure 3.8 the PBM node used to implement a single neuron is shown.

Let us consider a neuron that receives strong shunting inhibition that is at poisson times relieved for a short amount of time. During that short open window the membrane of the neuron will see the current influx through other activated ion channels and will evaluate, whether their combined conductance is high enough to overcome its spiking threshold; in other words, it measures whether it has received higher input to the inhibitory or excitatory 'mode' (with a bias to the inhibitory one). In the latter case, the neuron sends a spike to all downstream connected neurons that gets multiplied by the synaptic weight; this spike may or may not have an effect on the downstream neuron, depending on whether it is relieved of its gating inhibition within the PSP time constant ($\tau_e$ or $\tau_i$). The fact that each spike may have no effect at all, is modelled in the PBMnet by the Bernoulli probability ($p = \tau_e/\tau_G$) of ignoring messages from a given source. The former case, in which no output (spike) is produced, can be incorporated into a PBMnet simply making all outgoing connections of one node state have zero weight.

We saw in the previous section that the temporal correlations induced by a non-

poissonian spike distribution (namely oscillatory gating inhibition) was of minor importance. It can reasonably be expected that this generalizes to various other kinds of temporally correlated distributions.

Two facts complicate this picture somewhat and deviate from the PBM model.

1. In order to fit the previous analysis, the post synaptic potentials would need to have a simple square shape (since I simply counted numbers of messages) and extend for a sufficiently long time.

2. A neuron can in principle spike more than once, if it receives a very high input; unless the 'short open window' is very short indeed (of the order of magnitude of milliseconds).

The first issue can be addressed, by allowing connection weights to be random variables other than Bernoulli variables. I do not know of any theoretical analysis of this; in practice e.g. an RBM works also with weights as random variables (unpublished own work). If we assume some fixed post-synaptic kernel and a short integration window (during which the neuron's gating inhibition is released and it integrates its inputs on to the membrane potential) the effective weight the incoming spike will have, is simply the integral over the post-synaptic kernel for the duration of the time window; to obtain a probabilistic weight assume (as previously) that the presynaptic spike time and the release of inhibition occur at random time points.

The second issue, I find more interesting. Firstly it could be dismissed by assuming a neuron model whose refractory time constant is longer than the spike window opened by intermittently inactive shunting inhibition. However, the possibility of spiking more than once makes the network more 'expressive' (there is a greater number of possible states) and could have important effects. Namely if the single neuron has a linear transfer-function, this would probably lead to a linear response with Gaussian noise in the limit of a high fan-in, low leak neuron (note that the time-scale for the leak would be given by the length of the 'short open window', so that low leak is in practice very easily achieved). A linear transfer function with Gaussian noise has been used in ANNs to obtain models with continuous valued variables. I have not verified in simulation that spiking neurons with oscillatory inhibition can indeed implement such a Gaussian linear model, to some extent I will address this analytically in the next chapter however.

## 3.11 Discussion

In this chapter I introduced the network model I will be using in a large part of this thesis; this model will prove its usefulness further in later chapters. For now I can say that it seems to be a good functional model of rhythmic inhibition in attractor networks. It introduces

a different way of doing neural sampling [FBOL10] and is to some extent an alternative to [BBNM11].

When modelling the functional effect of gamma rhythms a key development would be the prediction of physically measurable results. Such a prediction is made based on a model of perceptual bi-stability in the appendix B of this thesis.

Chapter 4

# Learning and Inference in PBMnets

Section 4.2 is partly based on the paper "Rhythmic inhibition allows neural networks to search for maximally consistent states" by Hesham Mostafa, Lorenz K. Müller and Giacomo Indiveri [MMI15b].

## 4.1 Introduction

In this chapter I will study in greater depth how a PBMnet can be taught to represent probability distributions. I look at alternative methods of modelling the distribution that is represented by a PBMnet. This allows me to relate PBMnets to models whose training is well understood. Next to allowing me to encode distributions from which samples are available by training the PBMnet, this also elucidates the relationship of the PBMnet to various other kinds of artificial neural networks among them McCulloch-Pitts neurons [MP43], RBMs [Smo86] and autoencoders [KM14].

## 4.2 PBMnet Connectivity and Encoded Distribution

In section 1.2 I described a method to derive the limiting probability distribution of an MCMC sampler. The methods for doing so that I mentioned are valid, but practically not used very often. The reason for this is that with the given construction there is no simple, 'closed-form' link between the transition matrix $T$ and the resulting $\pi_T$.

Practically one commonly makes use of the fact that for a given $\pi_T$ there are many $T$ that give rise to it and one is free to impose additional restrictions. The most widespread restriction is to require *detailed balance*:

$$T_{ij}p_i = T_{ji}p_j \tag{4.1}$$

Using this restriction has the key advantage that $T$ becomes very easy to construct for a

given $\pi_T$.

Unfortunately the PBMnets described previously do not give rise to a transition matrix that fulfils the detailed balance condition in general. Thus the standard way of finding a connection matrix that produces a given probability distribution is not viable for PBMnets.

Another approach of relating the probability distribution of a network to its connection structure is to check whether it falls under the category of Markov Random Fields (MRF). Every MRF with strictly positive probability density is also a Gibbs-Field (and vice-versa) for which it is known that the probability distributions it represents are always of the form

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_{c \in \mathsf{cl}} \Phi_c(x_c)\right), \tag{4.2}$$

[HC68].

Initially I thought that PBMnets are clearly MRFs: The requirement on an MRF is simply that each node's probabilities of taking certain values are independent of other node's values conditioned on its directly connected neighbours. In PBMnets in contrast each node's probabilities of taking certain values are independent of other node's values, if directly connected nodes are *clamped* to a certain value. Clamping and conditioning are however not necessarily the same.

'Clamping' a node means to set it to a certain value and as the system traverses its state space never letting that node take any other value. 'Conditioning' on a node on the other hand means running the network freely and after the run discarding all samples in which that node did not have a certain value. I confused the two, because in some systems they are equivalent, for example when the system is an MRF. In systems where the induced distribution does not factorize as in equation 4.2, there is a difference, because conditioning does not stop information flow between nodes that are not directly connected, while clamping does.

By measuring and comparing a clamped and a conditioned PBMnet one finds that PBMnets are in general not MRFs [Mos11]. However this does not exclude the possibility that some PBMnets with special structure or in a certain limit are MRFs.

Instead of continuing to work with the concept of the MRF I moved to directly examining how close to a Gibbs Field PBMnets behaved. This is convenient because a Gibbs Field can also be defined in terms of the behaviour of single nodes: If the probability of a node to take a value has e.g. the form of a sigmoid activation function or softmax of the weighted sum of the inputs, a network of such nodes will behave as a Gibbs field and the probability distribution represented by it will take the simple form of equation 4.2.

## 4.3 The PBM Activation Function for Binary Nodes

Any field of two-valued nodes whose constituent nodes take their values with probabilities given by a sigmoid activation function

$$\sigma(a) = \frac{1}{1 + \exp(-a)}, \tag{4.3}$$

where the activation $a$ is the weighted sum of inputs to a node,

$$a_i = \sum_j W_{ij} x_j, \tag{4.4}$$

is a Gibbs Field.

For nodes with more than two values, the sigmoid becomes a e.g. softmax (the sigmoid is the two-valued softmax). The sigmoid function (and to some extent the softmax) is further of interest because it is also used in algorithms like the multi-layer perceptron (MLP) or the restricted Boltzmann Machine (RBM) [Smo86]. In the RBM the activation is subsequently used as the probability of a Bernoulli experiment, so that the 'activity' $y = B(\sigma(a))$ takes values zero or one.

I will now examine the activation function of a PBM node in order to determine whether there are some regimes in which it will approximate the sigmoid function.

A possible behaviour PBM nodes could exhibit in order to function as Gibbs-Fields in a network is the softmax activation function

$$p(y = 1) = \frac{\exp(\alpha w_1^{\mathrm{T}} x)}{\exp(\alpha w_1^{\mathrm{T}} x) + \exp(\alpha w_0^{\mathrm{T}} x)} = \frac{1}{1 + \exp(\alpha (w_0 - w_1)^{\mathrm{T}} x)}. \tag{4.5}$$

Here $w_0$ and $w_1$ indicate the matrices corresponding to the connections weights going into the two states of the PBM node. The max function (or 'hard' WTA) that PBM nodes use produces output only where it receives maximal input; this could also be seen as the $\alpha \to \infty$ limit of the softmax (or in the language of Boltzmann Machines the $T \to 0$ limit). If we thus replace sigmoid units with hard WTA in a given network, they will perform like a zero temperature Boltzmann Machine, i.e. a Hopfield network [Hop82]. This is of limited use as such a system has no way of escaping local optima.

However we have not yet modelled the unreliable communication of PBMnets. The fact that some messages never have any effect on their destination allows the network to escape local optima: The connections that enforce them will intermittently become ineffective.

One interpretation of failed communication is to see it as instantaneous perturbations to the connection weights. Let us consider the effect of perturbed connectivity on a hard

WTA so that for a random perturbation we obtain

$$\begin{aligned} p(y=1) &= p\left(\lim_{\alpha\to\infty}\frac{1}{1+\exp(\alpha(w_0-w_1+\delta w)^T x)}=1\right) \\ &= p(((w_0+\delta_0)-(w_1+\delta_1))^T x>0) \\ &= p((\delta_0-\delta_1)^T x>(w_0-w_1)^T x) \end{aligned} \tag{4.6}$$

The distribution of the $\delta$s thus implicitly defines the distribution of $p(y=1)$. To match the definition of the PBM node in section 3.3 and to recover the Bernoulli probability of having effective weight $W_{i,j,k(t),l(t)}$ or zero, we choose a simple distribution of $\delta_{0,1}$: Each entry of $\delta_i$ has a probability of $p_{\text{off}}$ to be equal to minus that entry in $w_i$, so that the perturbed weights $w_i+\delta_i$ are the unperturbed weights $w_i$ with some entries set to zero.

A compact solution for the probability can be obtained in the case of binary weights, i.e. $w_i^{kl}\in 0,W$,

$$p(y=1)=\sum_{j=0}^{N_1}\sum_{k=0}^{j-1}\binom{N_1}{j}\binom{N_2}{k}p_{\text{off}}^{N_1+N_2-k-j}(1-p_{\text{off}})^{j+k}, \tag{4.7}$$

where the number of sources that are sending messages to state $i$ is denoted as $N_i$. This expression has the advantage of being exact for the PBM node, however, the double sum makes it difficult to work with. A plot of this function for several values of $N$ can be found in figure 4.1.

Another way of approaching the calculation of $P(y=1)$ is to ask: What is the probability that state 1 receives more inputs than state 2? I will again denote the number of sources that are sending messages to state $i$ as $N_i$ and the number of messages received as $x_i$.

$$P(\text{i receives } x_i \text{ messages})=\binom{N_i}{x_i}p^{x_i}(1-p)^{N_i-x_i} \tag{4.8}$$

where $p=1-p_{\text{off}}$. This gives

$$P(x_1>x_2)=p\left(\binom{N_1}{x_1}p^{x_1}(1-p)^{N_1-x_1}>\binom{N_2}{x_2}p^{x_2}(1-p)^{N_2-x_2}\right). \tag{4.9}$$

This can be simplified substantially using the Gaussian approximation for the binomial distribution. This approximation is valid when both $N_i$ are large and $p$ is not very small (if $Np$ is fixed with $N$ large, a Poisson approximation would be appropriate).

$$\binom{N_i}{x_i}p^{x_i}(1-p)^{N_i-x_i}\approx\mathcal{N}\left(N_ip,N_ip(1-p)\right) \tag{4.10}$$

Then we can use

$$P(x_1>x_2)=P(x_1-x_2>0) \tag{4.11}$$

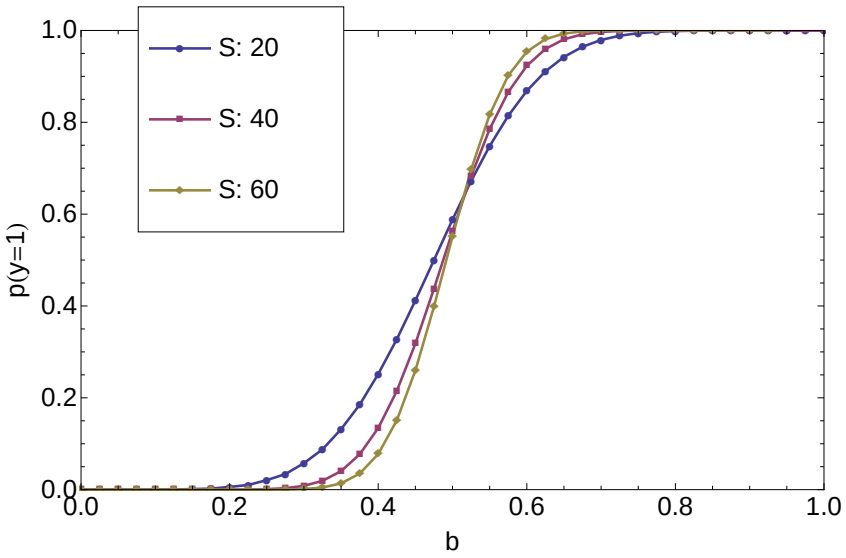



**Fig. 4.1:** *Activation probability of a PBM node for various input sizes $S$, $p_{on} = 0.5$ is fixed.*

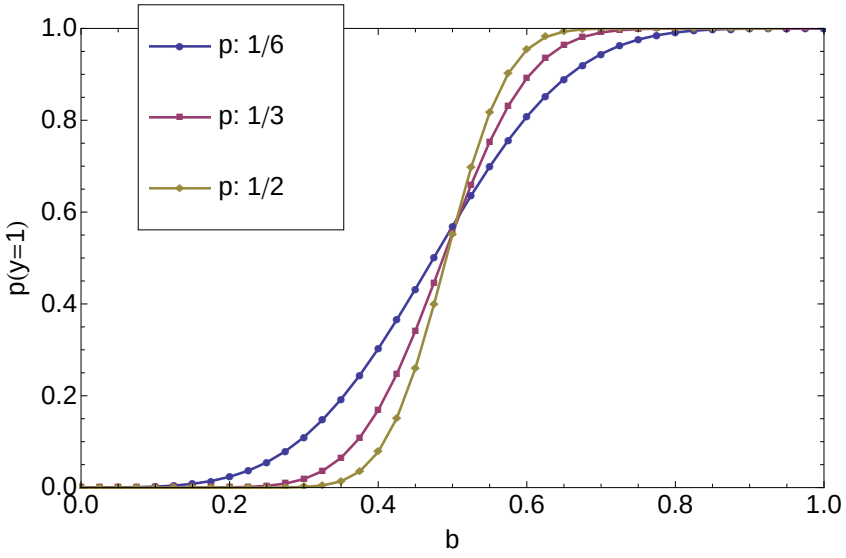



**Fig. 4.2:** *Activation probability of a PBM node for various transmission probabilities $p_{on}$, $S = 60$ is fixed.*

and that the difference of two Gaussian distributed variables is also Gaussian distributed

$$p(x_1 - x_2) = \mathcal{N}\left((\mu_1 - \mu_2),\sigma_1^2 + \sigma_2^2\right) \tag{4.12}$$

where $\mu_1 = N_1 p$, $\mu_2 = N_2 p$, $\sigma_1^2 = p(1-p)N_1$ and $\sigma_2^2 = p(1-p)N_2$. Then

$$P(x_1 > x_2) = P(x_1 - x_2 > 0) = P\left(\mathcal{N}\left((\mu_1 - \mu_2),\sigma_1^2 + \sigma_2^2\right) > 0\right). \tag{4.13}$$

The cumulative distribution function (CDF) of the Gaussian is the well-known error-function. With the parameters $S = N_1 + N_2$ and $\beta = N_1/S$ we obtain

$$P(y=1) = P(x_1 > x_2) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\sqrt{\frac{2Sp}{(1-p)}}\left(\beta - \frac{1}{2}\right)\right)\right]. \tag{4.14}$$

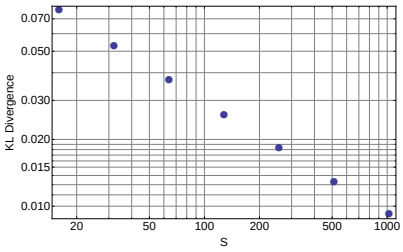
The slope of the function in Equation 4.14 and thus the approximate slope of the PBM node activation is

$$\left.\frac{dp(y=1)}{d\beta}\right|_{\beta=\frac{1}{2}} = \sqrt{\frac{2Sp}{\pi(1-p)}} \cdot \exp\left(\frac{-2Sp(\beta - \frac{1}{2})^2}{1-p}\right)\Bigg|_{\beta=\frac{1}{2}} = \sqrt{\frac{2Sp}{\pi(1-p)}} \tag{4.15}$$

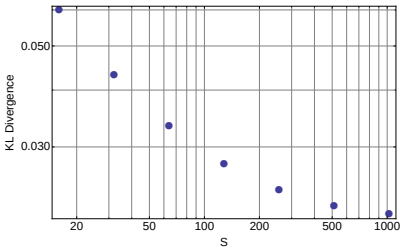
This result means that the slope of the activation function depends on $p$ for a given $S$; in other words a combination of $p$ and $S$ take a role analogous to the temperature of the Boltzmann Machine. This result is very useful for comparing the PBM activation function to a sigmoid: It elucidates how the temperature of the sigmoid used to fit the PBM activation function should scale asymptotically. One can then fit the sigmoid to have the appropriate temperature, by setting its slope at $\beta = 0.5$ equal to the slope of the PBM activation for the given $p$ and $S$. Alternatively one can set the $p$ to obtain a desired temperature given the fan-in $S$. Figure 4.3 shows the KL-divergence between the PBM activation function and the fitted sigmoid activation function as a function of $S$ with fixed $p$. Figure 4.4 shows a comparison of the exact PBM activation, the Gaussian approximation and a fitted sigmoid.

Notably the slope of this activation function changes with $S$ (unlike that of the RBM) and $S$ can in principle change between different samples: Since the 'zero' state of an RBM neuron is unconnected, only the active neurons in the presynaptic layer contribute to $S$. This problem however vanishes in the high fan-in limit: As the fan-in $L$ gets larger most presynaptic states correspond to an $S = \frac{L}{2}$ (this is the state of maximal entropy). As will be seen in later sections practically there indeed seems to be no problem with these slightly varying slopes / temperatures.

Another more general approach in which it is not necessary to make any assumptions about the distribution (or range) of the values in the connection matrix invokes the central

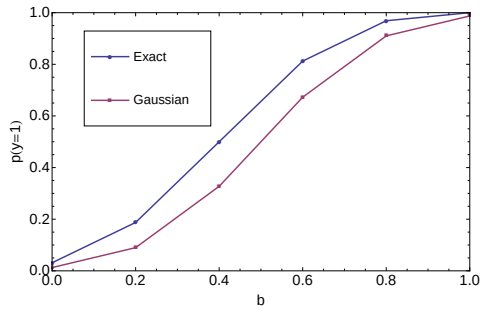



**(a)** *fixed p*

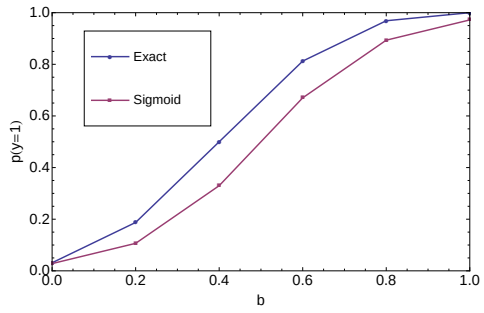



**(b)** *fixed 'Temperature'*

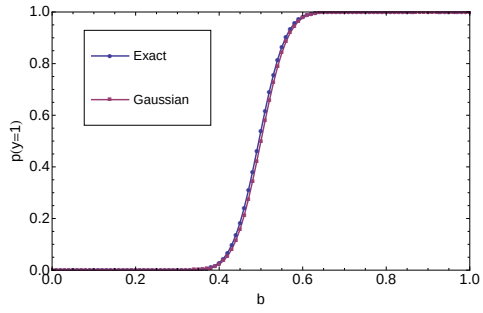
**Fig. 4.3:** *KL divergence between sigmoid and PBM activation as a function of $S$. Note the double-logarithmic scale: A straight line indicates a power law.*



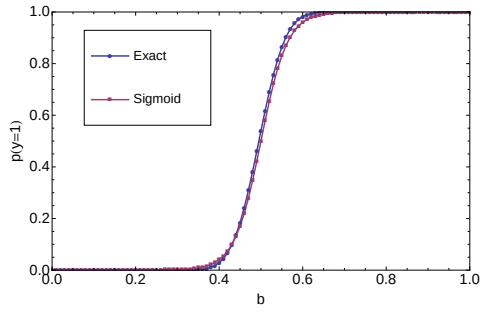

**(a)** *comparison to gaussian approximation from Equation 4.14, $S = 5$, $p = 0.5$*




**(b)** *comparison to slope fitted sigmoid, $S = 5$, $p = 0.5$*




**(c)** *comparison to gaussian approximation from Equation 4.14, $S = 100$, $p = 0.5$*




**(d)** *comparison to slope fitted sigmoid, $S = 100$, $p = 0.5$*

**Fig. 4.4:** *Activation probability of a oWTA node, compared to approximations*

limit theorem. Let us study the activation $a$:

$$a_i = \sum_j x_j w_{ji} B_j \tag{4.16}$$

where $x_j$ are the presynaptic activities, $w_{ij}$ is the connection weight and $B_j$ is a Bernoulli variable with probability $p_{\text{on}}$. Further we assume $x_j w_{ji} = q_j$ to be fixed; then we can write

$$a_i = \sum_j q_j B_j. \tag{4.17}$$

$a_i$ is now a sum over weighted Bernoulli trials. The central limit theorem states that the mean of a sufficiently large number of independent random variables is approximately normally distributed [BZ10] (if the random variables have well defined means and variances, which is clearly the case here). Formulaically this yields

$$a_i = \sum_j q_j B_j \approx \mathcal{N}\left(\langle p_{\text{on}} q_i \rangle_i, \langle p_{\text{on}}(1 - p_{\text{on}}) q_i \rangle_i\right), \tag{4.18}$$

where the angular brackets denote an average.

Again the probability of having an activity of one is the probability that this variable $a$ is greater than zero; to evaluate it I compute again the CDF of this Gaussian, which yields the same form as above. However in this approach there is no equally simple notion of the ratio parameter $\beta$.

This to some extent validates my intuition from section 3.10 that the activation is a linear response with Gaussian noise, however the variance of the noise increases with increasing mean.

### Discussion

Overall the observations in this subsection give the impression that the PBM function for large fan ins (a few tens), is a good approximation to the sigmoid. Thus PBM nets with high fan-ins behave approximately like Gibbs-Fields. This has the benefit that clamping a PBM node results in the rest of the network evaluating approximately the conditional given the value of that clamped PBM node. Further it has the benefit that algorithms that use a sigmoid activation function might be compatible with PBMnets.

A Gaussian CDF as activation function has an interpretation other than that of an approximation to a sigmoid: The functionality of a single node can be understood as a classifier that in its probabilistic classification decisions matches the probability ratio of two prototype signals corrupted by Gaussian noise. In fact this line of reasoning has been used to justify the use of sigmoid activation functions in neural networks (combined with the small numerical difference between the Gaussian CDF and sigmoids) [DJ99]. Due to

this interpretation I will refer to the mean-field PBM equivalent of an RBM as a 'Restricted Gauss Machine' (RGM).

## 4.4 The PBM Activation Function for Higher Order Nodes

For PBM nodes with more than two states the analysis becomes more complicated; define $m_i$ as the number of inputs that connect to $i$. Each of these has a fixed probability $p$ of successfully communicating with $i$ (these are now independent events). Then

$$P(\text{i receives } x_i \text{ messages}) = \binom{m_i}{x_i} p^{x_i} (1-p)^{m_i - x_i} \tag{4.19}$$

and

$$P(x_i > x_j) = p\left( \binom{m_i}{x_i} p^{x_i} (1-p)^{m_i - x_i} > \binom{m_j}{x_j} p^{x_j} (1-p)^{m_j - x_j} \right) \tag{4.20}$$

so that

$$P(\text{argmax}_i(x_i) = x_a) = \prod_{j \neq a} P(x_a > x_j); \tag{4.21}$$

using the same derivation as eq. 4.14 and $S_{a,j} = N_a + N_j$ and $R_{a,j} = N_a / S_{a,j}$ this becomes

$$P(\text{argmax}_i(x_i) = x_a) = \prod_{j \neq a} \frac{1}{2} \left[ 1 + \text{erf}\left( \sqrt{\frac{2(S_{a,j})p}{(1-p)}} \left( R_{a,j} - \frac{1}{2} \right) \right) \right]. \tag{4.22}$$

Again an argument for arbitrary weights can be made using the central limit theorem.

## 4.5 Learning in PBM Networks

Besides being able to evaluate approximate conditionals by clamping, the fact that a PBM-net in the high fan-in limit approximates an MRF has another major advantage: For some particular graph structures, there exist MRF learning algorithms that make it possible to find a connectivity structure that gives rise to a particular desired probability distribution of which samples are available.

The RBM [Smo86] is one such MRF that fortunately also fulfils the high fan-in criterion. The RBM connectivity structure is a bipartite graph, whose two disjoint subsets of vertices are called the visible and hidden layers. Learning in such a graph is achieved by 'Contrastive Divergence' (CD) [Hin02]. CD approximates the log-likelihood gradient by a locally (at each weight) computable quantity that could be seen as a sum of a Hebbian and an anti-Hebbian learning rule: The Hebbian term is active when samples of the desired distribution are clamped at the visible layer, the anti-Hebbian term is active when the system is free running.

A standard RBM operates with high parallelism: All nodes inside a layer update simultaneously conditioned on a fixed activity of the other layer. It has been shown previously that event based versions of this operation, where this exact simultaneity is violated, are feasible and a reformulation of CD (event-based CD) has been derived [NDPKDC14].

A PBMnet can approximately implement an event-based RBM, with an error that decreases for larger networks. For the oWTA network there is an additional potential problem in this approximation: The RBM is a probabilistic algorithm (units decide probabilistically based on their weighted input sum, whether to be active or not), while the oWTA network can be modelled by a probabilistic MCMC sampler, but is not probabilistic. How problematic this is, is hard to know without simulation. In these applications the underlying driver of the sampler is of little interest; the question is whether the high-level behaviour (the distribution that is sampled and temporal correlations between samples) are the quantities of practical importance.

## 4.6 RBM and oWTA

In the following I will examine by simulation whether 1) the PBM activation function is sufficiently close to the sigmoid to allow PBMnets to be trained by CD and the RBM energy function and 2) whether the use of oscillatory inhibition as an indirect PRNG can be applied to PBMnets (yielding a spiking oWTA network) implementing RBMs without substantially degrading the quality of the samples produced by the network. – I will find that RBM training can indeed be applied, but will also suggest a better alternative based on autoencoder training. Further we will see that the replacement of the stochastic process with an oscillatory one does not degrade the performance of the PBMnet terribly (also in this context).

### 4.6.1 Discrete time RBM with max-function activation and failing transmission

For a first test of the feasibility of implementing an RBM as a PBMnet I 'abstracted away' one of the key differences between the two: RBMs are defined in discrete time with simultaneous updates of all neurons in a layer, while PBMnets run in continuous time or at least use non-simulatneous updates. A hybrid of the two is described in the following; the aim is to assess whether the activation function (max-function with bernoulli perturbed weights / gaussian CDF) underlying the PBMnet supports learning with CD.

The network layout is identical to an RBM (a visible and a hidden layer without within-layer connections); layers are activated in turns, all neurons in a layer activate

simultaneously. The activation function is different from the RBM. The RBM uses

$$x_i = B\left(1, \frac{1}{1+\exp(\sum_j w_{ji} x_j)}\right), \tag{4.23}$$

where $B(n,p)$ is the binomial distribution with number $n$ and probability $p$, $x_k$ is the activity of neuron $k$ and $w_{ij}$ is the synaptic weight from neuron $j$ to neuron $i$. In contrast the (pseudo-meanfield) simultaneous-update, discrete time PBMnet (or RGM) uses

$$x_i = B\left(1, \frac{1}{2}\mathrm{erf}\left(\sum_j w_{ji} x_j\right) + \frac{1}{2}\right) \tag{4.24}$$

and the approximate RGM (or aRGM) uses

$$x_i = \Theta\left(\sum_j w_{ji} x_j B(1, p_{\mathrm{on}})\right), \tag{4.25}$$

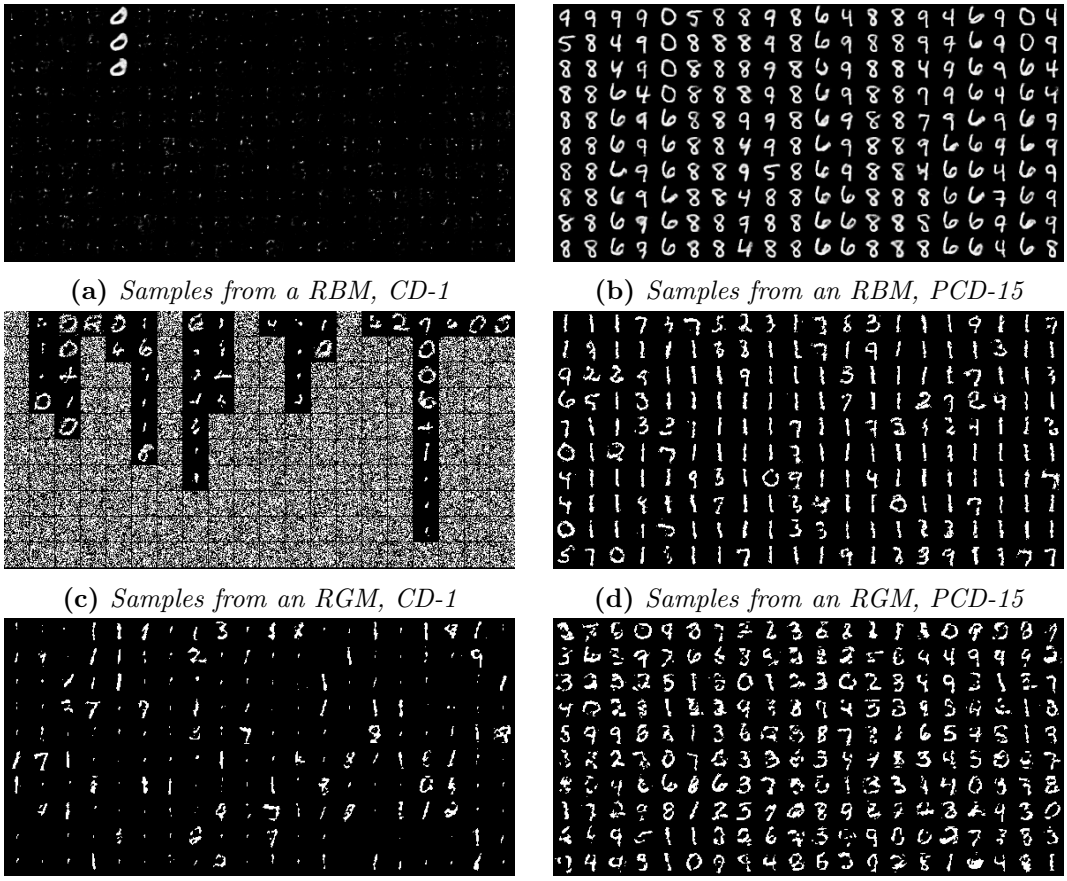
where $\Theta(\cdot)$ is the Heaviside step function and $p_{\mathrm{on}}$ is the transmission probability of the output spike of $x_j$. This function is very similar to the standard RBM function as has been exhaustively discussed in Section 4.3. In this notation it also becomes evident that equation 4.25 describes a McCulloch-Pitts neuron [MP43] with randomly failing spike propagation (or dropout [HSKSS12]).

Note that in transitioning from aRGM to RGM the scale of the weight acquires functional relevance; in the aRGM the 'temperature' is given by $p_{\mathrm{on}}$, while in the RGM it is given by the scale of the weight matrix. To translate weights from aRGM to RGM a scaling factor is necessary in general.

The 'communication failure' binomial does not need to be evaluated per weight, but only per unit; this saves a great amount of resources ($n$ rather than $n^2$ Bernoulli experiments). A hand-waving theoretical explanation might build on an assumption that the receptive fields of hidden units must be sufficiently different from each other; however if one assumes an over-complete representation in the hidden layer, this should introduce undesired dependencies in the hidden layer activation. In experiments the additional independence seemed not to have any impact.

Figure 4.5 shows samples produced by a RGM, aRGM and RBM with 500 hidden units and $p_{\mathrm{on}} = 0.5$; all were trained on MNIST [Mni] using PCD-15 [Tie08] and CD-1 [Hin02]. The samples indicate clearly that the RGM learned something 'reasonable' from the contrastive divergence with RBM cost. Interestingly simple CD-1 works better on the aRGM. I speculate that this is due to the 'dropout' [HSKSS12] performed in the aRGM update.

Similarly the fact that the aRGM samples different digit types in a short amount of



**(a)** *Samples from a RBM, CD-1*

**(b)** *Samples from an RBM, PCD-15*

**(c)** *Samples from an RGM, CD-1*

**(d)** *Samples from an RGM, PCD-15*

**(e)** *Samples from an aRGM, CD-1*

**(f)** *Samples from an aRGM, PCD-15*

**Fig. 4.5:** *Comparison of samples from an RBM and a RGM and aRGM trained on MNIST using CD-1 and PCD-15 with the same parameters. Horizontally arranged are different seed values, vertically 1000 steps of gibbs sampling from the state above.*

time is probably due to the 'dropout' in its update.

The aRGM is computationally cheaper than an RBM: No sigmoids need to be evaluated and instead of needing a random bit with varying probability $p_i$ at each node, it only requires a random bit of global fixed probability $p_{\text{on}}$.

In higher order aRGM (where a unit needs to have maximal activation in a group, in addition to having positive activation to become active) the same trick that is used in RBMs with hidden soft-max units [Hin10] can be used: Treat each state of the soft-max as the activation of a standard sigmoid unit during training. This procedure again yields a good model of the MNIST data (see figure 4.7) for an aRGM in which the hidden units are WTA units with six populations.

Notably this multi-population WTA hidden layer performs even better under training with simple CD-1. It may be possible to exploit this in implementations of e.g. deep belief networks.

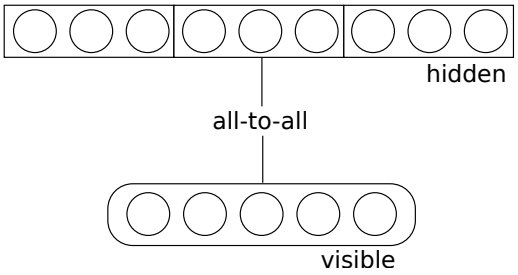
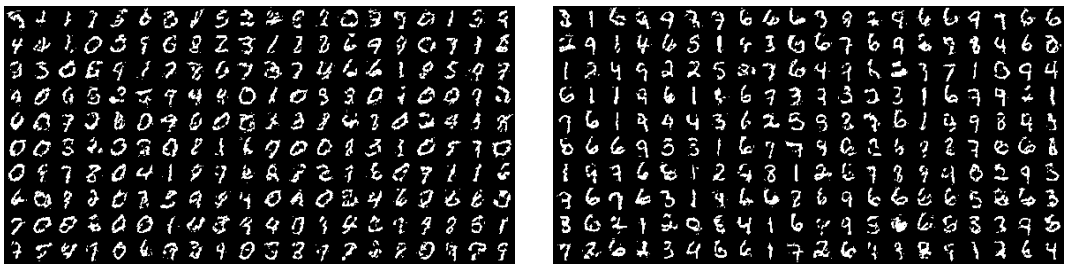



**Fig. 4.6:** *The layout of the higher-order aRGM. The visible layer functions as in the aRGM, the hidden layer is comprised of multiple higher order WTA (square boxes). For a hidden neuron to become active its input must exceed zero and the inputs to each of the other neurons in the same WTA.*



**(a)** *Samples from an aRGM, CD-1* **(b)** *Samples from an aRGM, PCD-15*

**Fig. 4.7:** *Samples drawn from aRGMs with five populations in each of the 100 hidden units.*

### 4.6.2 Restricted Gauss Machine as a Dynamical System

The mean-field limit of the RGM can be given a more rigorous mathematical treatment using the methods of [KM14]. Using these techniques I will derive an energy function underlying the vector field induced by the weight matrix of an RGM. Then the RGM training can be formulated as forcing data samples to become fixed-points, or even sinks of the associated vector-field. Finally this treatment allows me to make some statements about what kinds of errors are induced by the use of the 'wrong' energy of the previous sections.

The basic idea of this approach is to look at the network as an autoencoder. A given weight matrix is interpreted as inducing a vector field on the input sample space: Define the vector at each point in the input space as the vector between that point and its reconstruction under the current model. If we restrict this vector field to gradient fields it can be defined by an associated energy function.

To find the energy associated with the RGM, define the vector field

$$B(x) = \xi(r(x)) - \xi(x) \tag{4.26}$$

where $\xi(x)$ is the inverse of our activation function namely

$$\xi(x) = \text{erf}^{-1}(2x - 1) \tag{4.27}$$

and $r(x)$ is the hidden activation. The energy of the model is then

$$\int B(x)dx = \int \frac{1}{2}(1 + \text{erf}(x))\,dx + b^T x - \int \xi(x)dx. \tag{4.28}$$

Just like in the case of the RBM [KM14] the final term vanishes for binary data:

$$\int \xi(x)dx = \frac{1}{-2\sqrt{\pi}} \exp\left(-\text{erf}^{-1}\left((2x-1)^2\right)\right) \tag{4.29}$$

so that in the case of zero biases we obtain the energy function

$$\int B(x)dx = \int \frac{1}{2}(1 + \text{erf}(x))\,dx = \frac{1}{2}\left(u \cdot \text{erf}(u) + \pi^{-\frac{1}{2}} \exp(-u^2) + u\right). \tag{4.30}$$

This energy function can be used to train the RGM so that samples become fixed points of the dynamics; i.e. we minimize the difference in energy between a data-sample and the k gibbs steps distant sample (the analogue of contrastive divergence). Optionally we add the 'contractive regularizer'

$$\sum_k \exp\left(-(W_k x)^2\right) ||W_k||^2, \tag{4.31}$$

which ensures that the given data samples become sinks, rather than just fixed points of the dynamics [KM14].

The difference between the RBM energy and this RGM energy is that in the latter small errors incur a greater energy difference; in other words by mistakenly using the RBM energy to train an RGM model, I was more permissive of small errors (or relatively speaking less permissive of large errors).

Generally I see no fundamental problem with using any monotonic function of the energy instead of the energy to train the model: The same minima should occur in its energy-landscape. In fact it may be beneficial to deform the energy monotonically to achieve a different landscape around the minima (anecdotally I have seen better performance by scaling energies logarithmically or by the square root).

### 4.6.3 oWTA RBM

The final difference between oWTA and RBM that remains to be bridged, is that oWTA nodes are driven by individual oscillators, while RBM layers update simultaneously. To show that this difference is not crucial, I simulated an event-based oWTA network using

weight obtained by training an aRGM and observed the 'samples' produced by this network.

In this section I use a slightly modified oWTA node, one that is more suitable for hardware implementation and less biologically realistic: At every oscillator phase-reset, the oWTA takes a new state and communicates this state (with probabilistic failure possibility) as an event message to connected oWTA. The communication failures are modelled by simply not transmitting every second spike a neuron produces (a crude version of a 50% failure probability).

I show two different kinds of outputs produced by this oWTA network in figure 4.8: Firstly, in accordance with my definition of CTS, samples, which are snapshots of the states of all nodes at one time point. Secondly I weight the samples according to the total input the corresponding neuron received during this activity cycle; in an RBM this would correspond to the 'activation' or the probability of the neuron firing. Here there is on exact probabilistic interpretation of this quantity, however it represents in some way the confidence that this neuron should fire.

### 4.6.4 Discussion

The fact that this abstracted oWTA network can learn using a contrastive divergence scheme is remarkable because this can be expressed in terms of locally computable quantities. It is very plausible that a continuous time oWTA network could be trained using eCD [NDPKDC14]. This as well as the derivation of an eCD for arbitrary autoencoders would be interesting avenues for further research.

## 4.7 Deep Belief Network from RGM

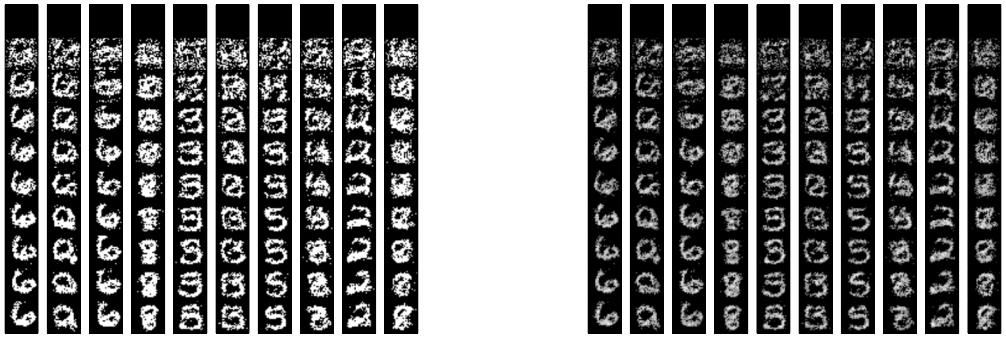
### 4.7.1 Synchronous Network

I tested the RGM model further by incorporating it into a Deep Belief Network [HOT06]. A DBN is essentially a multi-layer perceptron, that is made suitable for a deeper architecture. A key problem with deep architectures are vanishing gradients [Hoc91]. These are overcome in the DBN by initializing the weights by an unsupervised pre-training stage in which each layer is treated as an RBM. I classify MNIST using a stack of layers of sizes (784, 1200, 1200) (see figure 4.9) that are pre-trained as RGMs layer-wise; then for fine-tuning by back-propagation [RHW86] a 10-way soft-max (or logistic regression) layer is added on top (this is the RGM equivalent of a RBM DBN).

I implemented the synchronous DBN in theano [BBBLPDTWFB10].

As table 4.1 shows using the energy function derived for the RGM during pretraining improves the classification performance significantly.

The advantage of verifying my approach in a DBN is that for classifiers a clear measure of performance is available. In the generative models I studied before there is no simple



**(a)** *Spikes in visible layer* **(b)** *Weighted spikes*

**Fig. 4.8:** *oWTA aRGM samples; each column represents a new initialization at a state where the visible layer is inactive and the hidden layer is in a random state; each row is the state of the network after each neuron updated twice on average from the image in the preceding row. After a short burn-in the network produces MNIST-like digits.*

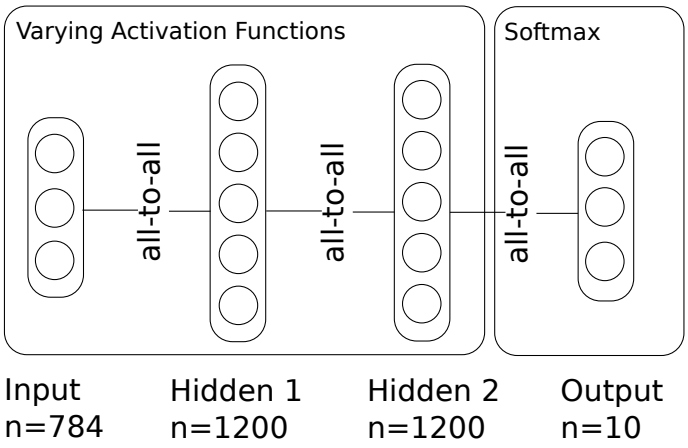



**Fig. 4.9:** *DBN Layout*

| Hidden Layers | Pre-Training Energy | k | Error |
|---|---|---|---|
| (1200,1200) | auto | 1 | 1.65% |
| (1200,1200) | auto | 3 | 1.50% |
| (1200,1200) | free | 1 | 2.25% |
| (1200,1200) | free | 3 | 1.99% |

**Table 4.1:** *Performance comparison; 'auto' refers to the energy in equation 4.30, 'free' to the standard free-energy of an RBM. k stands for the number of Gibbs sampling steps performed during CD pretraining.*

measure of performance; or rather the sensible measure of performance is intractable: For RBMs (and many similar models) the likelihood of a given datum is computationally intractable.

### 4.7.2 Asynchronous Network

The RGM-DBN can be implemented efficiently in an 'event' or 'spike' based manner; so that the mean-field behaviour of the implementation recovers the model of section 4.7.

I construct a network of nodes in the style of a PBMnet with the following properties: The connection weights between nodes are the same as given by the trained DBN from section 4.7. Every node 'updates' at predetermined points in time; the update order must be somehow 'randomized'. On updating the node sends out a spike, iff both its internal 'membrane' counter and 'spike-loss' counter are above a respective threshold. Incoming spikes have the effect of adding the connection weight to connected nodes' membrane counters (overflow behaviour is to stop at the ceiling) and outgoing spikes to increment the local 'spike-loss' counter by one (overflow behaviour is to roll over).

The second-to-last layer adds up its outgoing spikes over the full runtime instead of sending them; the logistic regression on top of the stacked RBM then uses these spike-counts divided by the average number of node updates as its input for classification.

I tested two methods for generating the randomized update sequences: The 'oWTA' method, where each node has a local oscillator that provides the update times. With an FPGA implementation in mind I propose the following alternate scheme: At every odd time step generate a random mask (one random bit per node) and update all the nodes with state 1 in this mask (in a fixed order); at every even time step invert the mask of the previous time step and update those with state 1 in the inverted mask (in a fixed order).

I achieve a classification performance of 2.4% with this scheme on MNIST in a simulated FPGA system and 2.5% with a simulated oscillator driven system; better results may well be possible with more parameter tuning and higher spike numbers. The results are summarized in table 4.2; they are comparable to (though slightly worse than) other recent event-based implementations [SNPGFL15; DNBCLP].

The performance gains from using the energy of equation 4.30 does not carry over into the event-based model. I speculate that this is due to the fact that the CDF of gaussian activation function (RGM activation function) is only an approximation of the true activation function of the event based simulation. It would be interesting to check whether an energy function derived with the same formalism as in section 4.6.2 for the exact activation would lead to even better performance (the integral required for this could be computed numerically and the values tabulated).

| Hidden Layers | Pre-Training Energy | k | Events | Update Order | Error |
|---|---|---|---|---|---|
| (1200,1200) | auto | 1 | 169 | masked | 5.5% |
| (1200,1200) | auto | 1 | 431 | masked | 3.8% |
| (1200,1200) | auto | 1 | 1740 | masked | 3.2% |
| (1200,1200) | auto | 1 | 168 | oscillatory | 6.9% |
| (1200,1200) | auto | 1 | 424 | oscillatory | 3.6% |
| (1200,1200) | auto | 1 | 1746 | oscillatory | 3.1% |
| (1200,1200) | auto | 1 | $\infty$ | synchronous | 1.65% |
| (1200,1200) | auto | 3 | 173 | masked | 4.8% |
| (1200,1200) | auto | 3 | 1740 | masked | 2.5% |
| (1200,1200) | auto | 3 | 178 | oscillatory | 8.4% |
| (1200,1200) | auto | 3 | 1737 | oscillatory | 2.9% |
| (1200,1200) | auto | 3 | $\infty$ | synchronous | 1.50% |
| (1200,1200) | free | 1 | 169 | masked | 5.7% |
| (1200,1200) | free | 1 | 1740 | masked | 2.9% |
| (1200,1200) | free | 1 | 168 | oscillatory | 12.7% |
| (1200,1200) | free | 1 | 1746 | oscillatory | 3% |
| (1200,1200) | free | 1 | $\infty$ | synchronous | 2.25% |
| (1200,1200) | free | 3 | 177 | masked | 4.7% |
| (1200,1200) | free | 3 | 1740 | masked | 2.4% |
| (1200,1200) | free | 3 | 180 | oscillatory | 8.9% |
| (1200,1200) | free | 3 | 1757 | oscillatory | 2.5% |
| (1200,1200) | free | 3 | $\infty$ | synchronous | 1.99% |

**Table 4.2:** *Performance comparison; 'auto' refers to the energy in equation 4.30, 'free' to the standard free-energy of an RBM. k stands for the number of Gibbs sampling steps performed during CD pretraining.*

## 4.8 Discussion

In this chapter I showed that the PBMnet, a model conceived to explain the effect of gamma inhibition lends itself to implementing deep belief networks. However although this DBN builds on a biologically inspired model, it is still very far from a biologically plausible model: The training of the DBN was done largely by backpropagation through several layers; a method for which it is difficult to imagine a biological counterpart. The same DBN trained with eCD [NDPKDC14] is somewhat more plausible as a model of biology; but even then there are great differences to biology, like the limited connectivity structure (no within layer connections). In short one should not read into this chapter the suggestion that the brain works like ANNs, but merely that PBMnets and oscillatory WTA can support some ANN versions.

Many difficulties in this chapter arose from the fact that the MCMC sampler induced by the PBMnet does in general not obey detailed balance. This poses the question: Can it obey detailed balance for some special cases? By studying equation 3.9 we find that this would be the case when there are always equally many inputs to both (or all) possible states of a node. This would happen in a high entropy state with random connections, but even then the time evolution of the network is such as to move away from this disordered state. It may still be possible to induce a detailed balance case, by not setting each term of the sum in equation 3.9 equal for symmetric entries in the transition matrix, but equating the full sums. I have not studied this possibility, since it is much more complex.

As I mentioned previously, through modifications to the operation of the single node different activation functions can be implemented (e.g. the linear activation with Gaussian noise mentioned in section 3.10).

It is interesting to see that other very recent spike-based implementations of deep networks [DNBCLP; SNPGFL15], reach a similar performance. Detailed comparisons in simulation and analysis might shed more light on this. The distinguishing feature of the models discussed here is that they can operate in a computationally cheap pseudo-probabilistic generative mode, unlike the aforementioned, which either are deterministic [DNBCLP] or use high-precision random number generators [SNPGFL15].

A historically interesting observation is that a McCulloch-Pitts neuron [MP43] from 1943 can fulfill about the same function as the conceptually more evolved sigmoid neuron from [HS83] (forty years later) by the simple addition of randomly failing spike propagation / synaptic transmission (yielding the aRGM neuron). The probabilistic interpretation of the sigmoid neuron is however not exactly applicable to the aRGM neuron, which profits from the more recent concept of the autoencoder.

A solid probabilistic interpretation of the aRGM might be given based on [GG15] where the probabilistic interpretation of dropout in various networks is elucidated.

Chapter 5

# PBMnets in Hardware

Section 5.2 is based on the paper 'Rounding Methods for Neural Networks with Low Resolution Synaptic Weights' by Lorenz Müller and Giacomo Indiveri [MI15].

## 5.1 Introduction

In this chapter I study practical problems arising in hardware implementations of ANN related algorithms. I first address the problem of limited synaptic resolution extensively and then discuss a framework for practical hardware implementations of deep belief networks.

## 5.2 Rounding Methods for Neural Networks with Low Resolution Synaptic Weights

Seeing that the results on oWTA networks pointed towards a potentially highly efficient method of implementing artificial neural networks (ANNs), I turned to a next problem that is relevant in this context: Low resolution synaptic weights.

Neural network algorithms simulated on standard computing platforms typically make use of high resolution weights, with floating-point notation. However, for dedicated hardware implementations of such algorithms, fixed-point synaptic weights with low resolution are preferable. The basic approach of reducing the resolution of the weights in these algorithms by standard rounding methods incurs drastic losses in performance. To reduce the resolution in the extreme case even to binary weights, more advanced techniques are necessary.

In this section I propose two methods for mapping neural network algorithms with high resolution weights to corresponding algorithms that work with low resolution weights and demonstrate that their performance is substantially better than standard rounding. I further use these methods to investigate the performance of four common neural network

algorithms under fixed memory size of the weight matrix with different weight resolutions and show that dedicated hardware systems, whose technology dictates very low weight resolutions (be they electronic or biological) could in principle implement the studied algorithms.

### 5.2.1 Context

Mapping floating point algorithms to fixed point hardware is a non trivial process. The choice of mapping method can have a major impact on the performance of the fixed point system. Standard neural network algorithms typically operate on floating point parameters when simulated on conventional hardware [AHS85; RHM+86; LM11]. Special purpose hardware (such as FPGAs and neuromorphic chips) on the other hand commonly implement synapses with fixed point resolution and possibly a small number of bits per synaptic weight [PWDFSERSM12; CBDDSCFD03; Ind02; PCDGPTB11; MAAICSAJIGN+14]. How this kind of hardware can best implement neural network algorithms is an open question.

The highly related question of how biological neural networks function under limited synaptic resolution has attracted significant attention in the neuroscience community. It has been argued that limited synaptic resolution has profound effects on the learning capacity of networks that use them [AF94; FDA05; LG13; BR08]. This calls into question whether it is plausible to think of the algorithms performed by biological neurons as equivalent to artificial neural nets (ANN). This analogy particularly applies to deep ANNs simulated with high resolution synaptic weights, which have been shown to be highly predictive of neural responses in visual cortex [YHCSSD14].

In the computational neuroscience domain, a method for using low resolution synapses is presented in [BL14], in which a spiking neural network is trained using an STDP learning rule. However [BL14] is only applicable to one specific learning rule and algorithm, a version of expectation-maximisation. In contrast I propose methods that work for several common neural network algorithms among them both discriminative and generative models.

In the integer programming domain a method called Randomized rounding (RR) [RT87] has been shown to be effective in online gradient descent on the convex problem of logistic regression; in this case an upper bound on the cost introduced by RR can be given [SGY13]. I apply the same method and other methods to neural network algorithms and also address the problem of the resolution of rounding probabilities.

[CBD14] examines the impact of low resolution synapses in deep learning architectures. [CBD14] focuses on different representations of low precision numbers (fixed point and floating point with different allocations of bits) with standard rounding, rather than algorithmic methods that intrinsically require lower resolution, as I will. These two approaches may well be complementary and yield best results when combined.

Finally the very recent paper [SNPGFL15] presents an effective method for weight

resolution reduction in a spike-based Deep Belief Network. In contrast to my approach their method necessitates the storage of a high resolution matrix during training (which is only necessary in some of the methods I discuss).

### 5.2.2 Mapping Methods

The two methods proposed for mapping continuous weight algorithms to low resolution ones differ in the way they update the synaptic weights of the neural networks: the first method is based on randomized rounding and works "online" in the sense that it changes the update procedure of the gradient descent at each update step; the second method is based on k-means and is an "offline" method, as it compresses a learned weight matrix after training.

The benchmark that these algorithms are tested against is based on the most straightforward technique of resolution reduction: normal rounding. For this benchmark we implemented a variant of gradient descent where at each time step the weight updates are rounded to fall onto values that are resolvable at the desired resolution. We refer to this method as *online rounding*.

#### Rounding

The first method I propose is used online, during training. It is makes use of the randomized rounding function: a function that maps a point in a continuous one dimensional space to a point on a discrete subspace. Specifically it maps it probabilistically to either the nearest point, or the second nearest point in the discrete subspace, with a probability that is inversely proportional to the distance to the corresponding point.

**Algorithm 1** Randomized Rounding

**procedure** RR($a$,$\epsilon$) ▷ $a$ mapped to $\epsilon$-grid
  $s \leftarrow sign(a)$
  $p \leftarrow \frac{|a|}{\epsilon} - \lfloor \frac{|a|}{\epsilon} \rfloor$ ▷ probability to increase abs. val.
  **if** $p > random(0,1)$ **then**
    $a \leftarrow s \cdot \epsilon \lceil \frac{|a|}{\epsilon} \rceil$ ▷ higher abs. val. grid point
  **else**
    $a \leftarrow s \cdot \epsilon \lfloor \frac{|a|}{\epsilon} \rfloor$ ▷ lower abs. val. grid point
  **end if**
  **return** $a$
**end procedure**

In the above $\lfloor \cdot \rfloor$ denotes the floor- and $\lceil \cdot \rceil$ denotes the ceiling function.

This randomized rounding method is applied during the gradient descent update. The update step then looks as follows.

I apply randomized rounding whenever a synaptic weight gets updated: Instead of

**Algorithm 2** RR Gradient Descent

1: **procedure** UPDATE($\theta$,$d\theta$,$\eta$,$\epsilon$) ▷ randomized rounding gradient descent, $\theta$: parameter, $d\theta$: gradient, $\eta$: learning rate, $\epsilon$: grid spacing
2: $\theta \leftarrow \mathrm{RR}(\theta - \eta \cdot d\theta, \epsilon)$
3: $\theta \leftarrow clip(\theta, -1, 1)$ ▷ clip $\theta$ to allowed range
4: **return** $\theta$
5: **end procedure**

being updated to a 32-bit floating point value, it gets updated to grid points $x_d \in [-\alpha,\alpha]$ with spacing $\epsilon$ ( i.e. $x_d \in \{n \cdot \epsilon \cap [-1,1] | n \in \mathbb{N}\}$). Where $\epsilon$ is chosen so that $2^i - 1$ grid points are available in total. I call this the *online stochastic method* with $i$ bits in the following plots.

Since in a hardware implementation the resolution of the probability in the RR procedure might be critical, I also ran this method with limited resolutions in $p$ (the resolution of $p$ was set equal to the resolution of the weights). The resolution of $p$ was reduced by standard rounding. I refer to this as the *coarse p method* in the following.

The proposed weights further get clipped to some permitted range $[-\alpha,\alpha]$ where $\alpha$ is an additional hyperparamter of the model. Unless otherwise indicated I set $\alpha$ to one in the following; best results are probably obtained with different values.

#### K-Means

In this method I first train the neural network with high-resolution parameters, and then use a technique taken from image compression (based on the k-means algorithm [Mac+67]) to extract k mean weight intensities. After clustering, the value of each pixel is set to the value of the center of the cluster it belongs to. In this offline method the full weight resolution is needed during training. In principle the clustering procedure could also be applied at every step of gradient descent, which would yield an online method in some sense, but compared to RR k-means is very expensive computationally and needs 'non-local' information.

This method requires additional storage for the cluster centre values so that the memory requirement is increased by $k \cdot \log_2(p)$, where $p$ is the precision of the center value. Note that this does not scale with the matrix size $n^2$ and is negligible for $n^2 \gg k$. Since this is the regime I are interested in, I will neglect this term in the following. I will refer to this method as *offline k-means*.

### 5.2.3 Results

I applied the aforementioned mapping methods to four types of neural networks: Multilayer perceptron (MLP) [RHM+86], restricted Boltzmann Machine (RBM) [AHS85], neural autoregressive distribution encoder (NADE) [LM11] and the maxout network [GWFMCB13b].

For all of these I investigated the impact of varying the parameter resolution under constant hidden layer size and, for the MLP and NADE, under constant weight matrix memory (scaling the resolution by a factor of $\beta$ also scales the size of the hidden layer by $1/\beta$).

The minimal resolution I consider is a 2-bit one, because these algorithms need at least three different values, a positive one, a negative one and zero. In neuromorphic hardware this can translate to two species of synpases (excitatory and inhibitory) with binary weights.

In the case of the RBM, it is difficult to give a scalar measure of performance, because the log-likelihood of some given data under a known RBM model is computationally intractable (unless the RBM is very small). To obtain a scalar measure for the performance of a generative model I applied our methods also to the NADE, an RBM inspired distribution learner of similar power, for which the log-likelihood assigned to some given data is tractable [LM11]. To assess the performance of the RBM, samples and connection weights produced in the different conditions are plotted.

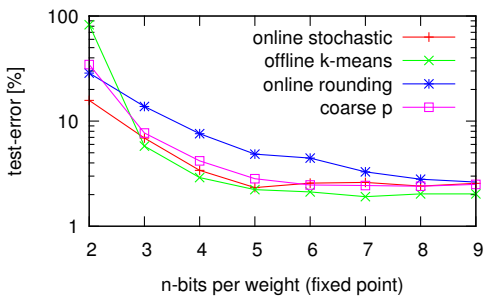
The MLP and RBM were trained on a binarized version of the MNIST hand-written digits dataset [Mni] in a theano-based [BBBLPDTWFB10] GPU implementation of batch gradient descent. The performance measure for the MLP is the percentage of the test-set samples that were misclassified.

The NADE was trained on the "dna" dataset from the libsvm webpage [CL11] using the code provided in the supplementary materials of [LM11] modified to allow our rounding methods. The performance measure for the NADE is the negative log-likelihood of the test set.

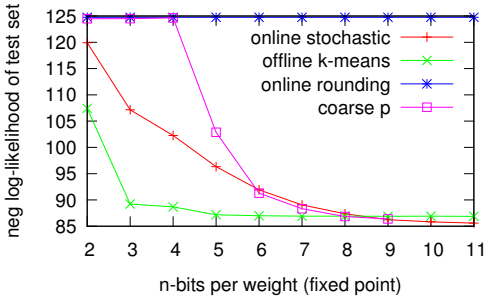
Figures 5.1a and 5.1b show how the performance changes as I increase the weight resolution under fixed hidden layer size (500 units). I observe that even 2-bit weights can perform far above chance level and I see a monotonically improving performance with higher resolution and a decrease of the performance gain per added resolution bit ending in a plateau, whose floor lies near the performance of the standard gradient descent performance. The location of the plateau floor indicates a slightly poorer performance of the low resolution algorithm; this is expected, because the low resolution algorithm cannot resolve continuous parameter values so that in the end phase of the descent it will randomly jump around the minimum rather than reaching it.

The 'coarse p' method, which is equivalent to normal rounding for 2-bit resolutions, surprisingly performs far above chance in the MLP even for 3-bit resolution.

The learning curves for a low resolution MLP (500 hidden units, 2-bit resolution) in Figure 5.2 show that for very low resolutions the model performs very similarly on the training as on the cross-validation set. This indicates that this model is limited by its expressive power, rather than by the learning algorithm (it has 'high bias' rather than 'high variance'). In light of this randomized rounding can also be interpreted as a regularization procedure.




**(a)** *Standard gradient descent with 32-bit floating point weights reached 1.81.*




**(b)** *Standard gradient descent with 32-bit floating point weights reached 84.6.*

**Fig. 5.1:** *5.1a Performance of MLP on MNIST, 5.1b Performance of NADE on 'dna' set*

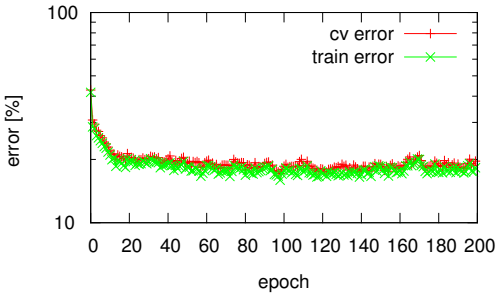
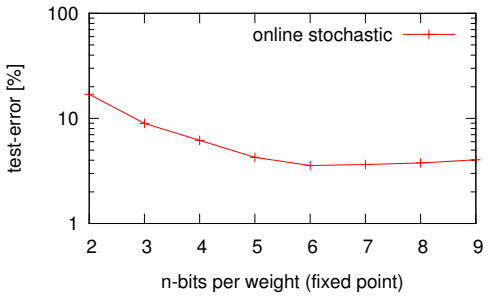
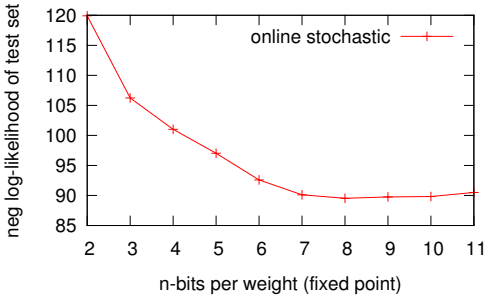



**Fig. 5.2:** *The cross-validation and training error are very close for a 2-bit resolution MLP on MNIST. This indicates a high-bias model.*

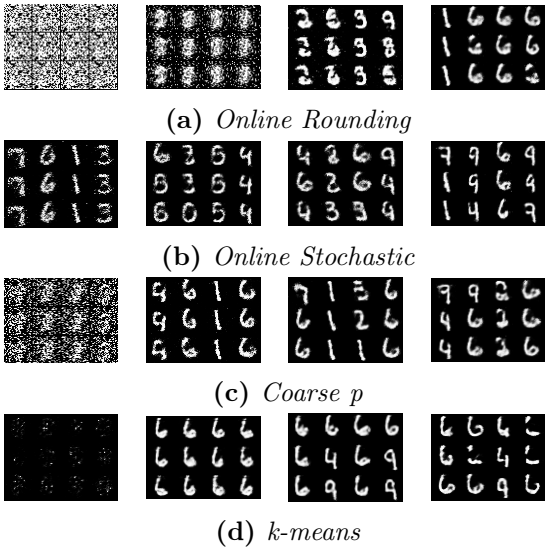



**(a)** *Optimal performance lies at 6.*




**(b)** *Optimal performance lies at 8.*

**Fig. 5.3:** *5.3a Performance of MLP on MNIST, 5.3b Performance of NADE on 'dna' set. Both under fixed matrix memory size.*



**(a)** *Online Rounding*

**(b)** *Online Stochastic*

**(c)** *Coarse p*

**(d)** *k-means*

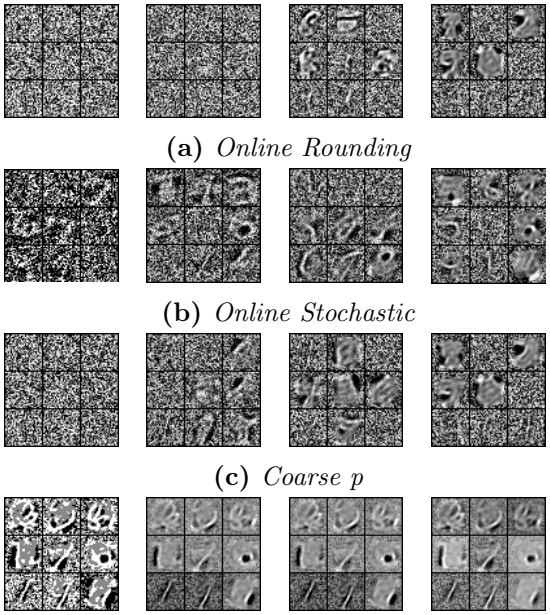
**Fig. 5.4:** *Samples (activations, not binarized) for 2, 4, 6, and 8-bit RBM trained with the four different resolution reduction methods. Inside each picture: Four different initial conditions (random test sample) horizontally arranged, over 3000 passes, printed every 1000 steps vertically arranged.*

Figures 5.3a and 5.3b show how the performances of NADE and MLP change as I increase the weight resolution while keeping the memory size of the weight matrix fixed at 400 bits. Under these conditions it is clearly preferable to choose an intermediate resolution.

Figure 5.4 shows activation probabilities for samples given by RBMs with different weight-resolutions (all have hidden layer size 500) trained with PCD-15 [Tie08]. As with the other algorithms the quality improves with higher resolutions, but even 2-bit weights already result in clearly recognisable digits (albeit noisy ones) for the randomized rounding method.

Figure 5.5 shows receptive fields learned in RBMs with varying weight resolutions. Notably there are some hidden units whose receptive fields 'look' very noisy for low resolution weights. However, it may well be the case that it is difficult to judge by eye what constitutes a 'useful' receptive field; conversely the weights for the 2-bit k-means method 'look' useful but do not produce good samples.

Finally I compare the online stochastic and online rounding methods on the maxout network from [GWFMCB13b] classifying CIFAR-10 [KH09] images without data augmentation to ensure the online stochastic method also works with deeper networks and more complex datasets (although we are targeting low precision rather than state-of-the-art machine learning). In table 5.1 shows that randomized rounding leads to much better performance than standard rounding also in this application. For the more complex CIFAR-10 dataset higher resolution weights were required than for the MNIST classification (MNIST on maxout yielded similar results to those observed in MLP).



**(a)** *Online Rounding*

**(b)** *Online Stochastic*

**(c)** *Coarse p*

**(d)** *k-means*

**Fig. 5.5:** *Final receptive fields for 2, 4, 6, and 8-bit RBM*

| Rounding Method | Resolution (bits) | Best Performance |
|---|---|---|
| Online Rounding | 6 | chance ($> 89\%$) |
| Online Rounding | 8 | chance ($> 89\%$) |
| Online Rounding | 10 | chance ($> 89\%$) |
| Online Stochastic | 6 | chance ($> 89\%$) |
| Online Stochastic | 8 | 28.84% |
| Online Stochastic | 10 | 23.4% |

**Table 5.1:** *Performance in test error of maxout network on CIFAR-10.*

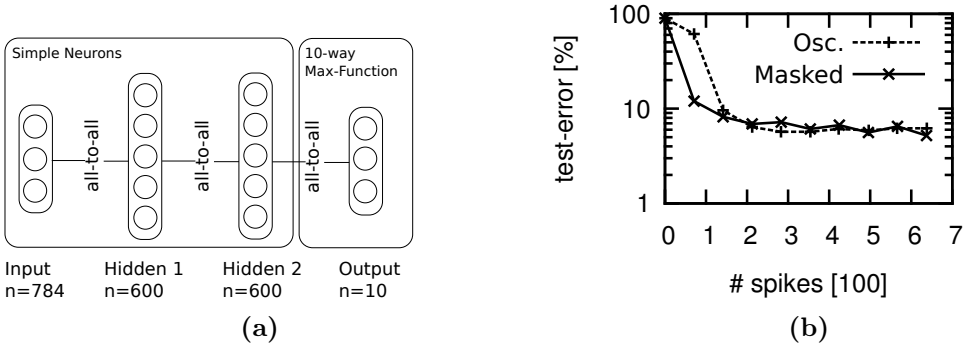



**Fig. 5.6:** *(a) Low resolution DBN Layout (b) Performance of the low resolution DBN as a function of the number of spikes it was allowed to produce. Asymptotic performance is around 5 percent test-error for both masked and oscillatory updates.*

## 5.3 A Hardware-Friendly DBN

As previously mentioned for practical implementations high-resolution floating point numbers are suboptimal. Here I show classification of an RGM-DBN (as in section 4.7) with reduced weight resolution (using the rounding schemes from section 5.2). This simulation takes into account the limited weight resolution at all points in the hidden layers (up to and *excluding* the output layer): Weights are limited to 2-bits and 'membrane potentials' to 5-bits. The limiting of the membrane potentials induces an additional deformation of the activation function that I could not describe analytically; for sufficient range of the membrane potential, this effect becomes negligible.

A formalized version of the update rules of the nodes described here and the setting in which they operate can be found in section 6.10.

## 5.4 Discussion

The methods outlined in this chapter give rise to a classifier that is relatively easily mappable to FPGAs and some neuromorphic chips. Arguably the probabilistic element of the DBN may not be practically useful for a classifier. This could be overcome by 'annealing away' during training the non-zero temperature of the model, yielding essentially a max-out unit whose output is limited to 1 bit.

A particularly interesting application of randomized rounding gradient descent, would be a neuromorphic neural network implementation with memristive synpapses that exhibit probabilistic switching [MRPCAPW11]. For other algorithms it has already been proposed that this behaviour could be exploited in neuromorphic hardware [BL14]. Thus it could be possible to implement the randomized rounding step directly in the memory unit, without need for a random number generator.

In this chapter I mostly considered the case of 'offline' learning (in the sense of not on-chip learning). A on-chip classification architecture would to my mind work best as

an unsupervised learning algorithm that preprocesses data for classification by e.g. a logistic regression. eCD [NDPKDC14] could be a good starting point to develop such an unsupervised learner, but in general good unsupervised learning is mostly an unsolved problem.

Chapter 6

# Stochastic Local Search in Event-based Networks

Section 6.1.1 is based on the paper "Recurrent Networks of coupled WTA-oscillators for solving constraint satisfaction problems" by Hesham Mostafa, Lorenz K. Müller and Giacomo Indiveri [MMI13]. Sections 6.10 and 6.11 are based on the paper "An event-based architecture for solving constraint satisfaction problems" by Hesham Mostafa, Lorenz K. Müller and Giacomo Indiveri [MMI15a] and parts of that paper are elaborations of other sections of this chapter.

## 6.1 Introduction

In [MMI13] we described a method to solve constraint satisfaction problems using networks of oWTA. Although I did not think of it in these terms at the time, the algorithm this network performs is a Stochastic Local Search (SLS) algorithm. In this section I will examine how some SLS algorithms for solving constraint satisfaction problems (CSPs) can be formulated as event-based networks and I will give a brief account of past and potential hardware implementations of these networks. I will start by looking at SAT problems for which a particularly powerful algorithm can be translated into an event-based network.

A key realization to seeing the close connection between the previous chapters of this thesis and the current one, is that Gibbs sampling or MCMC algorithms are in some sense SLS algorithms. Specifically one can construct a particular kind of SLS algorithm by constructing a Gibbs sampler that visits the states that the search ought to find with higher probability than others. However in the generalization away from just the mentioned samplers, lies great potential, due to a relaxation of the requirements on the path through the search/sample space taken: In a search scenario the algorithm does not need to visit various non-solutions with some specified probability, rather it ought to find the true

solution as quickly as possible.

### 6.1.1 oWTA for Solving Constraint Satisfaction Problems

In [MMI13] we used the same basic mechanisms to solve CSPs as described in section 3.6 (see figure 3.3; as mentioned in that section these networks are the work of Hesham Mostafa): WTAs whose attractor state is periodically destroyed are coupled together so that under favourable phase-relations they can force each other to change state so as to fulfil constraints encoded in their connectivity.

The network activity performs an SLS. Whenever a WTAs attractor is destroyed, the network re-evaluates one variable (this constitutes the 'small variation' of a proposal solution that is the hall-mark of SLS algorithms). The heuristic is 'greedy': The new value of the variable is the value whose corresponding population receives the highest input. This greediness is tempered by the fact that only inputs at favourable phase-relation are taken into account; this allows exploration away from the optimal state (given the neighbours) and ensures that only global optima are stable (a proof of the stability / instability of other states is given in [MMI15b]).

### 6.1.2 Hardware Architecture

The CSP networks from [MMI13] can be streamlined and generalized for hardware implementation in the following way, suggested by Hesham Mostafa [Mos11]. The functional units of the network are no longer WTAs but nodes communicating through asynchronous events. Each node contains an oscillator with particular frequency $f_i$ that differs from the other nodes' frequencies and a simple parametric model that specifies an input/output relationship. Whenever the phase of the associated oscillator reaches a multiple of $2\pi$, the node outputs an event based on its inputs and its parametric model. The connectivity of the nodes and their parameters determine the behaviour of the system.

### 6.1.3 This Chapter

The architecture and oWTA based CSP solver I just described seemed very promising ideas to me. However in the described form they cannot plausibly claim equating or even nearing the performance of a standard sequential architecture running state-of-the-art algorithms.

In this chapter I will formulate several powerful CSP solving algorithms for this architecture; these 'network' formulations of existing serial algorithms show the true potential of this platform, but are also suitable for other massively parallel hardware, as long as the connectivity can be implemented and the computing nodes exhibit sufficient complexity (a different method for randomizing update ordering has to be introduced then). Specifically implementations on parallel, synchronous hardware are in principle also possible.

In the following I conceptually take a step away from the standard approach of neuromorphic engineering: I no longer think of the individual nodes as parametric neuron models, but rather as programmable cores. While this does actually not change the definition of the networks I investigate (programming is a complex form of specifying parameters), this change in viewpoint makes reasoning about them easier.

## 6.2 SAT Problems

SAT problems are a particular kind of constraint satisfaction problems. The aim in solving a SAT problem is to find an assignment of truth values to a set of variables $l_i \in L$ so that all members of a set of constraints $c_j \in C$ (called 'clauses') are fulfilled simultaneously. These clauses have a particular form: They are or-conjunctions of $n$ negated and/or non-negated variables. A SAT problem in which the maximum number of variables in any clause is $n$ is called an n-SAT problem.

In formulas an e.g. 3-SAT problem with $m$ clauses can be written as follows.

$$c_i = (!_q l_q) \vee (!_r l_r) \vee (!_s l_s), \tag{6.1}$$

where $!_i$ is a negation or non-negation. Find $l_i$ so that

$$c_1 \wedge c_2 \wedge \cdots \wedge c_m = \text{True} \tag{6.2}$$

1-SAT can obviously be solved in $\mathcal{O}(m)$ steps. 2-SAT can also be solved in polynomial time, e.g. Papadimitriou's algorithm [Pap91] (see next section) is a probabilistic algorithm with $\mathcal{O}(m^2)$ runtime. 3-SAT and n-SAT for $n > 3$ are in general NP complete [Sip96].

The fact that $(n > 2)$-SAT problems are NP complete implies that the best known algorithms that solve them have exponential run time. In recent years it has been found that this does mean that it is impossible to find solutions to some high order and large SAT problems relatively quickly and SAT solvers have found practical use in various real world applications.

## 6.3 The Schöning-Papadimitrou Algorithm

Probably the most easy to understand algorithms for 3-SAT is Papadimitirou's algorithm [Pap91]. The idea behind this algorithm is simply to satisfy random unsatisfied clauses by randomly changing one of their variable's truth assignments.

Schöning proposed a minor modification to this algorithm that allowed him to derive a worst case upper bound for its convergence time [Sch99], namely $(2(1 - 1/k))^n$. This bound is one of the best theoretical bounds for any SAT solver (the algorithm however is in practice not state-of-the-art).

**Algorithm 3** Schöning-Papadimitriou Algorithm (SPA)

```
1: Input: Formula F, maxFlips
2: Output: satisfying assignment a or unsolved
3: a ← Randomly generated assignment
4: for i ← 1, maxFlips do
5:     if a is model for F then
6:         return a
7:     end if
8:     c_i ← randomly selected unsatisfied clause
9:     l ← random variable pertaining to c_i
10:    flip(l)                          ▷ flip that variable's truth value thereby fulfil c_i
11: end for
12: return unsolved                     ▷ if the algorithm gets here, the search was unsuccessful.
```

**Algorithm 4** Schöning's Algorithm

```
1: Input: Formula F, maxTries, maxFlips
2: Output: satisfying assignment a or unsolved
3: for j ← 1,maxTries do                          ▷ Difference to SPA: Restarts
4:     a ← Randomly generated assignment
5:     for i ← 1, maxFlips do          ▷ Difference to SPA: maxFlips is set to 3 · n_clauses
6:         if a is model for F then
7:             return a
8:         end if
9:         c_i ← randomly selected unsatisfied clause
10:        l ← random variable pertaining to c_i
11:        flip(l)                      ▷ flip that variable's truth value thereby fulfil c_i
12:    end for
13: end for
14: return unsolved                     ▷ if the algorithm gets here, the search was unsuccessful.
```

Since the two algorithms are almost identical and Schöning provides the theoretical analysis and explains extensions to other constraint satisfaction problems while Papadimitrou to my knowledge first formulated it, I refer to Alg. 3 as the Schöning-Papadimitrou algorithm (SPA).

### 6.3.1 Generalization to other CSP

SPA can be easily generalized to any CSP [Sch99]: Randomly changing the value of a (discrete) variable pertaining to a uniformly random, unfulfilled constraint is a generally viable strategy for unweighted CSP. The following section can be trivially extended to such cases. However, for general CSP I think it is quite intuitive that more complex heuristics deliver substantially better results.

## 6.4 Network Formulation of the Schöning-Papadimitrou Algorithm

In this section I will explain how the SPA can be approximately implemented by a network of entities similar to oWTA variables. For simplicity of formulation I will assume that an internal oscillator defines the pseudo-random update sequence of the nodes; however as in section 5.3 other methods for generating the pseudo-random update sequence could be used.

Each clause and each variable is assigned a node, as explained in section 6.1.2 each node contains an oscillator with particular frequency $f_i$ that differs from the other nodes' frequencies. Whenever the phase of the associated oscillator reaches a multiple of $2\pi$, the node updates and communicates. The update and communicate functions differ between variable and clause nodes. The connections between nodes are given by the SAT problem.

When a clause node updates, it checks whether it received any spikes (messages) in its state F terminal during its last cycle. If so, takes the value 'fulfilled' and does not communicate anything; if not and it received at least one spike in the state U terminal it takes the value 'unfulfilled' and communicates a spike to a random variable connected to it (this random assignment can be made dependent on the identity and order of recently received spikes, to save a local PRNG). If the clause did not receive any spikes, it stays at the value it had previously and repeats the corresponding action. When a variable updates, it checks whether it received any spikes during its last cycle. If so it takes the state that last received a spike; if not its state remains the same. Then the variable communicates a spike to all clauses connected to its currently active state. Figure 6.3 shows the network corresponding to a 3-SAT problem with a single constraint.

Clearly the only stable state of this network is the one where all clauses are fulfilled. If at least one clause is unfulfilled it will force one of its connected variables to flip its state

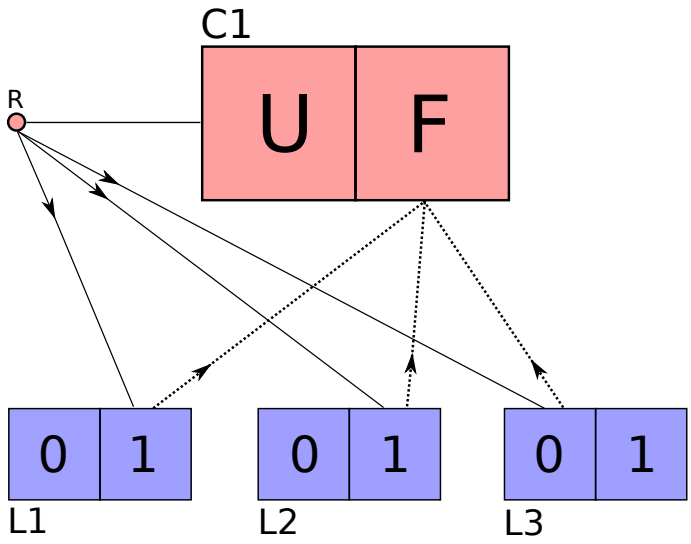



**Fig. 6.1:** *This is the network corresponding to the problem* $(L1 \vee L2 \vee \neg L3)$ *or alternatively the problem* $C1$ *solved by algorithm 3. When the* $C1$ *node updates and finds it is in state U (unfulfilled), it communicates a spike to a randomly selected (represented by R) variable (to the state that fulfils the clause). When the variables update, they take the state that last received a spike.*

during the next cycle (fulfilled clauses do not communicate, so they do not interrupt the message). On the other hand, if all clauses are fulfilled, they do not send spikes any more and the variables do not change their states, since they do not receive spikes.

Notably the frequencies $f_i$ need to be assigned such that for any clause node $c$ and any variable $l$ the relation $f_c < f_l$ holds. This ensures that during each clause node's cycle every variable connected to it is updated at least once. Then a fulfilled clause is guaranteed to receive at least one spike during its cycle; if $f_c > f_l$ this is not guaranteed.

This is not a perfect parallelization of Alg. 3; there are several discrepancies between it and the network behaviour. Firstly the choice of the next unfulfilled constraint that flips a variable arises from a deterministic function (phase resets of incommensurable oscillators); however most practical implementations of probabilistic algorithms use deterministic PRNGs. This discrepancy is almost a necessity; it may be aggravated by the fact that oscillators constitute a PRNG of potentially low quality. Secondly there are several information delays in the network: When a clause updates it does not know whether a different clause already instructed one of its connected variables to switch state, if that variable has not yet performed its update.

One situation in which this second 'error' may occur can easily be prevented however, namely the situation in which a clause node has not received any messages during its period, even though one of its connected variables has received a spike that would cause it to fulfil the clause (the clause would eventually be informed of this change through a spike, but it may by then already have forced another of its associated variables to change state unnecessarily). In the network this information lag can be overcome in two

ways: The clause node that causes the switch of a variable can also send a spike to all associated constraints that need the same variable state to put them into the 'fulfilled' state. Alternatively the variable nodes could send immediate spike messages, whenever they receive a switching input. The former method has the drawback of requiring additional connections, the latter requires variable nodes to spike outside their standard spike-window.

The opposite situation is more complex to prevent; when a clause has received a fulfilling message, but the associated variable is instructed to switch state before the clause updates. To prevent this error the clause needs to have the ability to keep track of the states that its associated variables represent (n bits of information in an n-SAT) rather than just whether any of them sent a message (1 bit of information). If the clause can keep track of the state of each associated variable, the second method mentioned above, can be adapted to prevent these errors.

In simulations I have observed only correcting both error types yields good results. All subsequent results use this method.

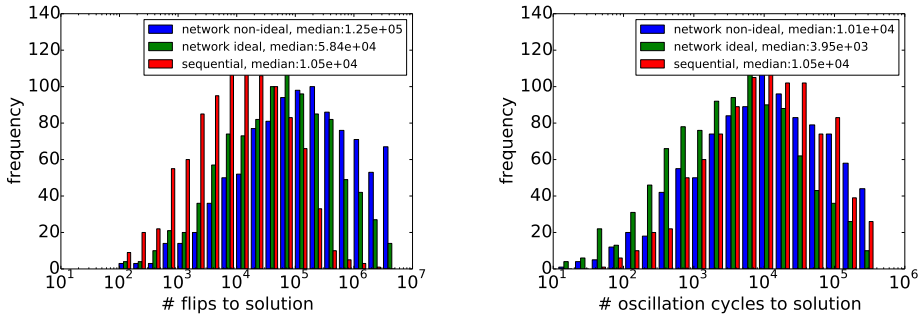
## 6.5 Network and Sequential Schöning-Papadimitrou: Performance Comparison

The figures in this section illustrate the performance differences between various network implementations and a sequential implementation of the Schöning-Papadimitrou algorithm (Alg. 3). The benchmarks are a set of intermediate size, difficult 3-SAT problems taken from [HS98], with 50 variables and 218 clauses.

The plots in Figure 6.2 show the performance of the network without the aforementioned possible errors for two different cases: The 'ideal' case where messages are transmitted instantly and never lost and a 'non-ideal' case where messages have a delay uniformly distributed between zero and ten percent of the node update cycle and a ten percent chance to get lost completely.

I plot two different measures of performance. For the sequential implementation the time to solution can be measured in an implementation independent way by the number of variable flips. For the network I also use show this measure, but additionally I plot the number of node oscillation cycles to solution: This second measure takes into account that the network can update nodes in parallel.

I interpret these two measures in the following way. The number of flips indicates whether the oscillators as PRNG had an impact on the algorithm. The number of oscillation cycles indicates how fast a hardware implementation would need to be to run faster than the standard algorithm on CPU (which achieves around 1-10 Mega-Flip per second). Since the average number of oscillation cycles to solution is about equal to the number of flips performed by the sequential algorithm, a hardware implementation (with imperfect routing) would need to run at 1-10 MHz to perform as well as a conventional computer.




**(a)** **(b)**

**Fig. 6.2:** *Comparison of performance of network and sequential Schöning-Papadimitrou algorithm. Note that the histogram has logarithmically spaced and sized bins.*

## 6.6 The probSAT Algorithm

The state-of-the-art SLS SAT solver called probSAT [BS12] is closely related to Algorithm 3. Since probSAT is much faster at solving large instances of SAT problems, I tried to formulate it as a network as well.

The key difference between probSAT (given in Alg. 5) and SPA is that unfulfilled clauses do not choose uniformly at random which associated variable to switch, but they use a 'make' and 'break' heuristic. 'Make' heuristics take into account how many clauses are newly fulfilled when a certain variable switches (this number will be denoted as $m$); 'break' heuristics count how many clauses are newly unfulfilled when a given variable switches ($b$). The exact from that this non-uniform random decision based on $m$ and $b$ takes, is given by a heuristic function $f(m,b)$.

The heuristic function $f(m,b)$ can take several different forms; which of these works best depends on the problem that is studied. The 'exponential' algorithm uses the form

$$f(m,b) = \frac{x^m}{y^b} \tag{6.3}$$

where $x$ and $y$ are parameters. The special case where $x = 1$ and $f(m,b)$ becomes independent of $m$ is the one that was used in the 2014 SATcompetition [BDHJ]. The other form the authors of [BS12] propose is

$$f(m,b) = \frac{m^x}{\epsilon + b^y} \tag{6.4}$$

where $x$,$y$ and $\epsilon$ are parameters. Note again the special case $x = 0$ for which $f(m,b)$ only depends on $b$ (which was also used in the SATcompetition). Notably probSAT won the parallel random track of that competition.

**Algorithm 5** probSAT

**Input:** Formula $F$, maxTries, maxFlips
**Output:** satisfying assignment $a$ or unsolved
**for** $j \leftarrow 1$,maxTries **do**
    $a \leftarrow$ Randomly generated assignment
    **for** $i \leftarrow 1$, maxFlips **do**
        **if** $a$ is model for $F$ **then**
            **return** a
        **end if**
        $c_i \leftarrow$ randomly selected unsatisfied clause
        **for** $l \in c_i$ **do** ▷ Difference to SPA: Evaluate a heuristic function
            Compute $f(m(l),b(l))$
        **end for**
        $l \leftarrow$ random variable of $c_i$ with probability $\frac{f(m(l),b(l))}{\sum_{k \in c_i} f(m(k),b(k))}$
        flip($l$) ▷ flip that variable's truth value thereby fulfil $c_i$
    **end for**
**end for**
**return** unsolved ▷ if the algorithm gets here, the search was unsuccessful.

## 6.7 Network Formulation of Break-only probSAT

The basic idea is the same as with the Schöning-Papadimitrou algorithm. Each clause and each variable is assigned a node. Each node contains an oscillator with particular frequency $f_i$ that differs slightly from the other nodes' frequencies and has two (or more) states (oscillation modes) it can take. Whenever the phase of the associated oscillator reaches a multiple of $2\pi$, the node updates and communicates. The update and communicate functions differ between variable and clause nodes. The connections between nodes are given by the SAT problem.

The difference lies in the update behaviour of the clause nodes. In addition to the previously mentioned operations, the clause node evaluates at phase reset, whether there currently is a single variable that could 'break' it, if that variable switched its state. If there exists such a potentially 'breaking' variable the clause node sends a special 'break' message to every constraint that the variable is connected to. Each constraint node accumulates these 'break' messages over its cycle. If during its update, a constraint node is required to flip a variable, its choice of variable will not be uniform any more, but influenced by the 'break' messages it received.

I evaluated two ways in which the break messages can influence the choice of node:

1. The constraint node chooses to flip the variable with which the fewest 'break' messages are associated.

2. The constraint node chooses at random which variable to flip, but if it is identical to the one with which the last 'break' message was associated, it chooses a different

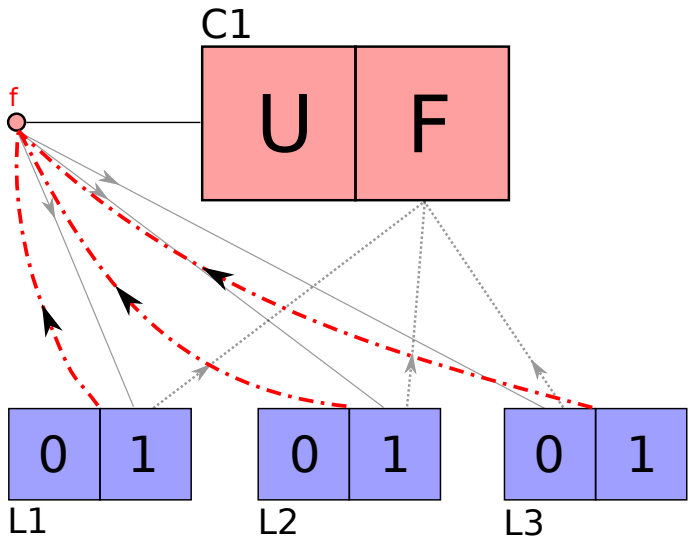



**Fig. 6.3:** *This is the network corresponding to the problem* $(L1 \vee L2 \vee \neg L3)$ *or alternatively the problem* $C1$ *solved by algorithm 5. When the* $C1$ *node updates and finds it is in state* $U$ *(unfulfilled), it communicates a spike to a heuristically selected (represented by* $f$*) variable (to the state that fulfils the clause). When the variables update, they take the state that last received a spike and send a 'break' message to* $f$ *if appropriate.*

one.

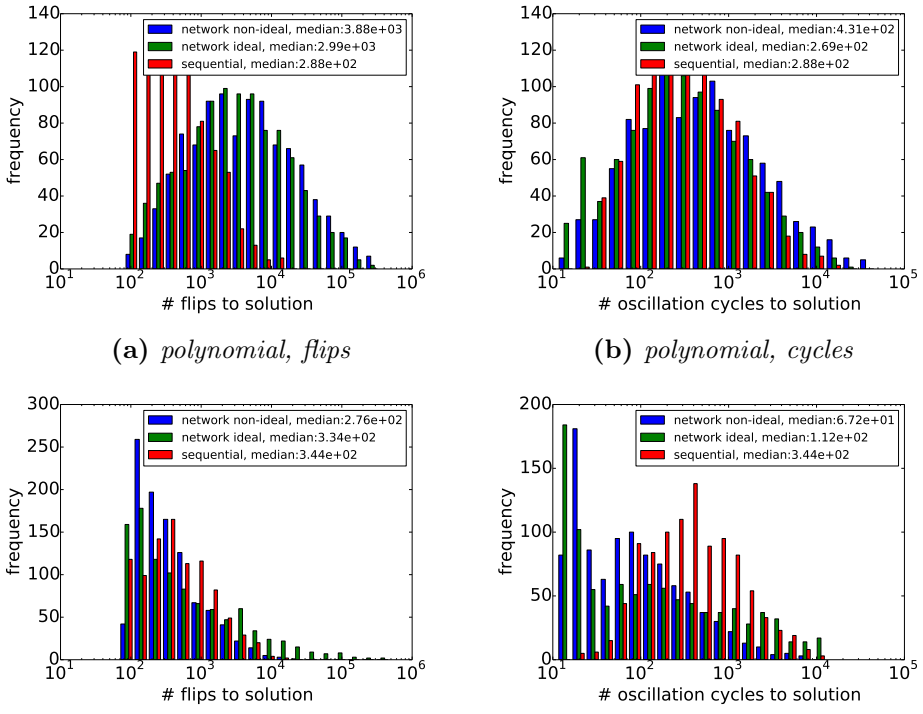
The first method is a very steep function of $b$, more reminiscent of the exponential criterion and becomes probabilistic only if the message passing is made unreliable (refer to section 4.1 for an in depth examination of this). The first method is much shallower in $b$.

## 6.8 Network and Sequential probSAT: Performance Comparison

The figures in this section illustrate the performance differences between various network implementations and a sequential implementation of the probSAT algorithm [BDHJ] (Alg. 5).The first set of benchmarks are a set of intermediate size, difficult 3-SAT problems taken from SATLIB [HS98], with 50 variables and 218 clauses.

The plots in Figure 6.4 show the performance of the network without the aforementioned possible errors for two different cases: The 'ideal' case where messages are transmitted instantly and never lost and a 'non-ideal' case where messages have a delay uniformly distributed between zero and ten percent of the node update cycle and a ten percent chance to get lost completely.

The quantities plotted are the same as in section 6.5. Again the hardware implementation would need to be clocked at 1-10$MHz$ to be as fast as a conventional computer. In this context this is more remarkable, because probSat is a state-of-the-art algorithm




**(a)** *polynomial, flips* **(b)** *polynomial, cycles*

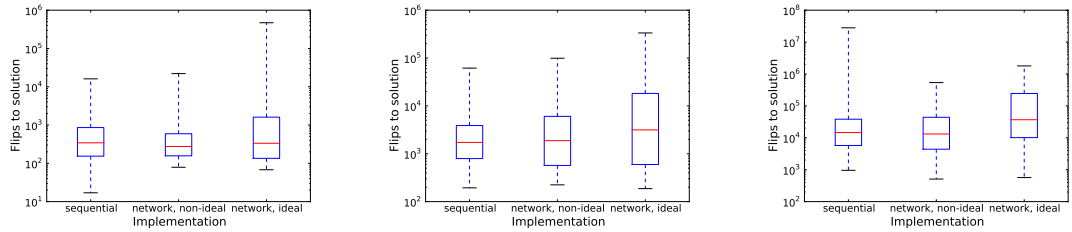
**(c)** *exponential, flips* **(d)** *exponential, cycles*

**Fig. 6.4:** *Comparison of performance of network and sequential probSAT algorithm with exponential and polynomial heuristic functions. Note that the histogram has logarithmically spaced and sized bins. The best performing algorithm in either metric is the network probSAT with routing errors and max-function heuristic. In plot 6.4d the distribution looks bimodal, because for very small numbers of cycles to solution, this number was not accurately measured (probabilistically upper bounded).*

for random SAT instances. The simple logic that the constraint and variable nodes need to execute could certainly run at such a speed. Another potential bottleneck is that on the order of $10^5$ messages need to be exchanged to solve one of these problems within $10^{-4}$ seconds to be comparably fast as a conventional computer. For this relatively small problem a giga-bit bus could thus supply sufficient bandwidth; but for larger problems a smarter routing structure is probably necessary.

I cannot give a comparison on present day state-of-the-art benchmarks, because the behavioural simulator of my network implementations runs the variables at circa $10Hz$ oscillation cycles on standard hardware and an estimated $10^6 - 10^9$ cycles are needed to solve a single state-of-the-art problem (i.e. 1-1000 days per problem). Because of this it is important to evaluate the performance on various problem sizes to ensure that the network probSAT scales as well with size as the sequential one; if this is the case, there is no reason I can think of, why the network would not perform equally well on larger problems (a test of this would most likely have to be done in a dedicated hardware implementation though).

The next benchmarks are also drawn from SATLIB, but are larger problems (100 and




**(a)** *50 variables, 218 constraints* **(b)** *100 variables, 430 constraints* **(c)** *200 variables, 860 constraints*

**Fig. 6.5:** *Comparison of performance of network and sequential probSAT algorithm with exponential heuristic function on different problem sizes. The red line indicates the median, the box outlines the 1st and 3rd quartile and the whiskers show the full range of the data. Sequential and network formulations scale equally well with problem size.*

200 variables, with 430 and 860 constraints). The network version seems to scale about equally well as the sequential one (Figure 6.5). Even the largest problems require passing less than $10^9$ binary messages which should certainly be possible for an on chip router in less than a second.

## 6.9 MaxSat Problems

MaxSat problems are a variation of k-Sat; the formal setup is the same but the goal is different: Rather than finding an assignment such that

$$c_1 \wedge c_2 \wedge \cdots \wedge c_m = \text{True} \tag{6.5}$$

find an assignment that maximises $\sum_i c_i$ the number of satisfied constraints.

MaxSat is less suitable to being solved by a network than k-Sat for several reasons. First of all in contrast to k-Sat the optimal solution to MaxSat is not stable under any SLS algorithm I can think of. Secondly in order to remember what configuration found so far was optimal a global observer would need to be introduced (in contrast to k-Sat there is no 'local' description of optima).

In practice there is still one constraint under which it might be useful to solve MaxSat in a network: Any time computing; i.e. if it is necessary to be able to get an approximate solution to the MaxSat problem at any time during the runtime. E.g. the network probSat implementation avoids very bad solutions altogether and implements an any time MaxSat solver.

## 6.10 Hardware Implementation

In the previous sections I have outlined 'network' formulations of two SAT solvers in natural language; however, in the architecture described in 6.1.2 the algorithms can also be formalized more stringently by defining the following network of nodes. Also this formalization outlined in this section is primarily the work of Hesham Mostafa. A node has $N$ externally accessible input ports on which it can receive events and $M$ output ports on which it can send out events. The special input port $in.0$ port receives events from an internal analogue oscillator. The digital logic has internal state variables, $\vec{s}$. On the arrival of an event (external or internal) and based on the index of the input port receiving the event and the current state of the digital logic, the digital logic evaluates the index of the output port to which it should send the event (The event is suppressed by sending it to the dummy $out.0$ port), updates its state $\vec{s}$, then sends out an event in that order. The algorithm the architecture runs is fully described by the event routing function G, the state update function F and the connections between nodes.

Note that this slightly generalizes the functionality outlined in 6.1.2: Nodes may now produce output events on any input event; I suggested adding this functionality to avoid errors as the ones described in section 6.4. A generalization that is useful for mapping algorithms to other architectures, is that the analogue oscillator can be replaced by any mechanism that ensures irregular update ordering of the nodes (e.g. the masking scheme proposed in section 4.7.2). The batching involved in this method however degrades the performance for some SLS algorithms.

## 6.11 Formalized Algorithms for Event-Based Hardware

### 6.11.1 Graph Colouring

Given this formal description of the architecture I can now easily describe new algorithms; e.g. for graph colouring (developed together with Hesham Mostafa).

A $k$-colouring for a graph $G$ with vertices $V(G)$ is a map $\phi : V(G) \rightarrow C$ where $C = \{x|x \in \mathbb{N}_0, x < k\}$. Given $E(G)$ the set of edges of $G$, a proper colouring of $G$ is a colouring $\phi$ such that $\phi(x) \neq \phi(y)$ for all $\{x,y\} \in E(G)$.

#### Algorithm and Network

An example algorithm suitable to this hardware for finding proper k-colourings of a given graph $G$ is the following (see Fig. 6.6). On an oscillator reset, the nodes switch the state of the boolean variable 'heuristic'. If since the last reset any events have arrived at the input port corresponding to the current color of the node, the node selects a new color; if 'heuristic' is true it chooses the color with the fewest conflicts (neighbours of the same color) otherwise it moves to the adjacent right color for some fixed color ordering. Then it

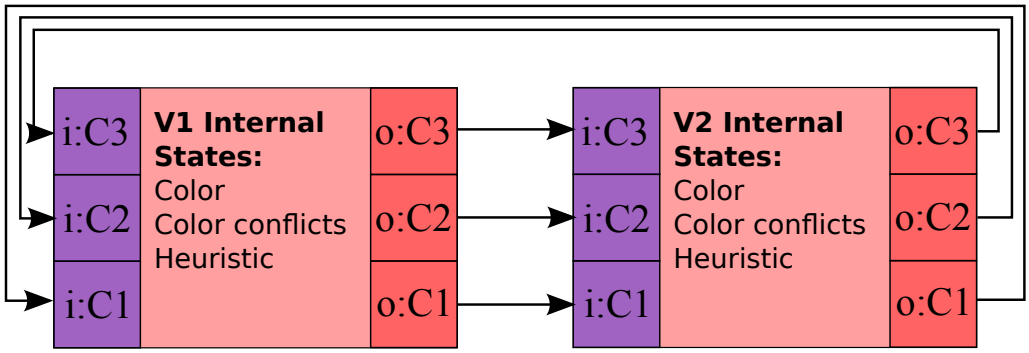



**Fig. 6.6:** *This is the network corresponding to the 3-coloring of the graph* $V = \{V1,V2\}$, $E = \{(V1,V2)\}$. *The squares at the edge of the box indicate input ports (purple) and output ports (red). Events are routed along the arrows. Spiking behaviour is described in the main text.*

resets the color conflict information. On other input events the node increments the color conflict internal state variable.

#### Network: Performance

I assessed the performance of this algorithm on several k-colouring problems of intermediate difficulty (see table 6.1) taken from [RR11] in which a different massively parallel coloring algorithm was assessed (called 'gravitational swarm intelligence' (GSI)). As in the case of boolean satisfiability, I cannot attempt state-of-the-art sized problems, since the software simulator of our hardware is $10^6 - 10^9$ times slower than a hardware implementation would be.

The algorithm is quite simple: A min conflict heuristic takes turns with a heuristic free update of conflicting nodes. In terms of numbers of cycles to solution this algorithm compares favourably to [RR11], see table 6.1.

### 6.11.2 Formalized descriptions of Algorithms

Here I describe the algorithms I introduced previously in a clear and concise 'pseudo-code' manner (I give the formalized network versions of probSat, k-colouring and spiking aRGM (DBN)). $\mathrm{G}(s,i)$ determines at which port an incoming event generates an output, $\mathrm{F}(s,i)$ describes the change of internal state variables.

This formalization makes it particularly clear how these different algorithms are supported by the same architecture. Further it demonstrates that a programmable version of this architecture could be a useful platform and what the programming interface would need to offer.

Finally one can envision a complier that translates such an algorithmic description to VHDL code for implementation on FPGA.

**Algorithm 6** k-colouring

**State Variables $\vec{s}$:** color; heuristic; $n_{c,1}$, $n_{c,2}$, ... , $n_{c,k}$
**Input Ports:** osc; $i_1$, $i_2$, ... , $i_k$
**Output Ports:** none; $o_1$, $o_2$, ... , $o_k$
**Connections:** $i_j$ to $o_j$ $\forall j$ between all adjacent nodes
**function** G($\vec{s}$,in)
    **if** in = osc **then**
        spike at port $o_{\text{color}}$ where 'color' is the state variable
    **else**
        spike at port none
    **end if**
**end function**
**function** F($\vec{s}$,in)
    **if** in = osc **then**
        heuristic← not heuristic
        **if** $\sum_i n_{c,i} > 0$ **then**
            **if** heuristic = True **then**
                color ← color with lowest $n_{c,i}$
            **else**
                color ← $mod$(color + 1,$k$) + 1
            **end if**
        **end if**
        $n_{c,i} \leftarrow 0$ $\forall i$
    **end if**
    **if** in = $i_j$ **then**
        $n_{c,j} \leftarrow n_{c,j} + 1$
    **end if**
**end function**

**Algorithm 7** aRGM

1: **State Variables $\vec{s}$:** sum, spike-loss, tieBreak
2: **Input Ports:** osc; $w_{i,\pm}$
3: **Output Ports:** none; $o$
4: **Connections:** $o$ connects on neighbouring neurons to port $w_{i,\pm}$ according to the weight of the trained model
5: **Parameters:** $T_1$, $T_2$; their choice defines the effective connection probability
6: **function** G($\vec{s}$,in)
7: **if** in = osc and spike-loss$< T_1$ and (sum $> 0$ or (sum $= 0$ and tieBreak)) **then**
8: spike at $o$
9: **else**
10: spike at none
11: **end if**
12: **end function**
13: **function** F($\vec{s}$,in)
14: **if** in = osc **then**
15: sum $\leftarrow 0$
16: tieBreak $\leftarrow$ False
17: spike-loss $\leftarrow$ mod(spike-loss+1,$T_2$)
18: **else if** in = $w_{i,\pm}$ **then**
19: sum $\leftarrow$ sum + $\pm i$
20: tieBreak $\leftarrow$ not tieBreak
21: **end if**
22: **end function**

**Algorithm 8** probSat Network: Constraint

**State Variables $\vec{s}$:** $b_1$, ..., $b_k$; $s_1$, ..., $s_k$; $f_1$, ..., $f_k$
**Input Ports:** osc; $s_j+$, $s_j-$; $i_{\text{inc},j}$ ($j$ runs from 1 to 3)
**Output Ports:** none; $o_j$; $o_{\text{inc},j}$ ($j$ runs from 1 to 3)
**Connections:** variable output $\pm$ maps to associated constraint input $s_j\pm$; constraint output $s_j f_j$ maps to the variable state that fulfils this constraint; constraints that share variables map $\text{inc}_j$ to $b_h$ inc (where $j$ and $h$ are the respective indices of the variable)
**function** G($\vec{s}$,in)
    **if** in = osc and $\forall j : s_j \neq f_j$ **then**
        spike at port $o_j$, choose $j$ so that $b_j$ is minimal
    **else if** in = $s_j\pm$ and $\sum_h (s_h = f_h) = 1$ and $s_j = f_j$ **then**
        spike at port $o_{\text{inc},j}$
    **else**
        spike at port none
    **end if**
**end function**
**function** F($\vec{s}$,in)
    **if** in = osc **then**
        $b_j \leftarrow 0\ \forall j$
    **else if** in = $s_j\pm$ **then**
        $s_j \leftarrow \pm$
    **else if** in = $i_{\text{inc},j}$ **then**
        $b_j \leftarrow b_j + 1$
    **end if**
**end function**

**Algorithm 9** probSat Network: Variable

**State Variables $\vec{s}$:** $s$
**Input Ports:** osc; $i_+$, $i_-$
**Output Ports:** none; $o_+$, $o_-$
**Connections:** See constraint
**function** G($\vec{s}$,in)
    **if** in = osc **then**
        spike at port $i_s$ where $s$ is the state variable
    **else if** in = $i_+$ **then**
        spike at port $o_+$
    **else**
        spike at port $o_-$
    **end if**
**end function**
**function** F($\vec{s}$,in)
    **if** in = $i_+$ **then**
        $s \leftarrow +$
    **else if** in = $i_-$ **then**
        $s \leftarrow -$
    **end if**
**end function**

| Graph | #vertices | #edges | Density | K | #GSI | #ours |
|---|---|---|---|---|---|---|
| myciel7 | 191 | 2360 | 0.13 | 8 | 302 | **145** |
| myciel6 | 95 | 755 | 0.17 | 7 | 92 | **31** |
| myciel5 | 47 | 236 | 0.21 | 6 | 97 | **19** |
| myciel4 | 23 | 71 | 0.28 | 5 | 25 | **3** |
| myciel3 | 11 | 20 | 0.36 | 4 | 21 | **2** |
| david | 87 | 986 | 0.21 | 11 | 208 | **95** |
| anna | 138 | 812 | 0.21 | 11 | 300 | **8** |
| huck | 74 | 662 | 0.22 | 11 | 84 | **8** |
| jean | 80 | 508 | 0.16 | 10 | 165 | **16** |
| queen 5x5 | 25 | 160 | 0.53 | 5 | **302** | ? |
| 1_fullins_3 | 30 | 100 | 0.23 | 4 | 37 | **11** |
| 1_fullins_4 | 93 | 593 | 0.14 | 5 | **76** | 366 |
| 1_fullins_5 | 282 | 3247 | 0.08 | 6 | **222** | 1593 |
| 2_fullins_3 | 52 | 201 | 0.15 | 5 | 67 | **47** |
| 2_fullins_4 | 212 | 1621 | 0.07 | 6 | 176 | **120** |
| miles_250 | 128 | 387 | 0.04 | 8 | **317** | 2021 |

**Table 6.1:** *Number of cycles to convergence on benchmarks given in [RR11] of our network algorithm and a different massively parallel algorithm (GSI). Each number in the ours column was averaged over 4 runs with redrawn oscillator frequencies; one run for the queens graph did not converge in* $10^5$ *cycles, the other runs averaged* 530 *steps to convergence.*

## 6.12 Discussion

The key technical result of this chapter is that I specify an algorithm (network probSAT) for which clear specifications a massively parallel, event-based implementation must meet, can be given so that that implementation could match current state-of-the-art approaches on standard hardware. As far as I am aware this is the first such result (for a somewhat practically useful algorithm).

This success stems mostly from the fact that probSAT is surprisingly well suited for a distributed implementation. Other SAT solvers, especially non-SLS solvers, would probably be quite difficult to implement and non-SLS solvers would even in the best case require some kind of global signals.

The full potential impact of the presented algorithm (and other algorithms of this thesis) becomes clearer in view of the recent work of Hesham Mostafa to build hardware of the kind described in 6.1.2. A first prototype exists and it has been shown that arguments about the ease of implementation of the mismatched oscillator paradigm are justified. The prototype however uses a serial off-chip router and a overly simple parametric node model; to achieve state-of-the-art performance a parallel on-chip router and more complex nodes are required (and under development).

An open question is the practical usefulness of a MaxSat variant of the discussed SAT

solvers. The development of an application for an any time MaxSat solver (as described in section 6.9) seems a challenge of its own to me. Perhaps control problems could be cast in such a framework to get very many fast approximate motor commands rather than a few optimal ones - but this is pure speculation.

Chapter 7

# Discussion and Conclusion

The algorithms I present in this thesis are best suited to the architecture we present in [MMI15a]. Hesham Mostafa is currently working on a physical implementation of this architecture. Based on my simulation results it seems realistic to me that the algorithms I propose in chapter 6 running on this architecture could surpass the current state-of-the-art in SAT solving. On the same architecture the algorithms proposed in chapters 4 and 5 could provide a low-power classification system and generative model learner.

The presented event-based algorithms are also amenable to simulation on other platforms. While an oscillation-based implementation of the PRNG seems to me the most practical and efficient, it is not the only one. The key requirement are a random update sequence and approximately equal update rate of all nodes. In simulation I demonstrated that an FPGA generated randomized update sequence by masking, while more expensive, works about equally well as the oscillatory system. Generally the networks described in this thesis should be amenable to implementation on FPGAs, SpiNNaker hardware [FGTP14], True-North [MAAICSAJIGN+14] and other neuromorphic systems. An implementation on FPGA of the algorithms I propose in chapters 4 to 6 is currently in development in our group and an implementation on True-North is being investigated by others.

Based on the detailed simulations in chapter 6, we have a clear idea of potential performance bottlenecks in an FPGA implementation of the proposed SAT solvers. Namely the single node behaviour can easily run at the required speed, but the number of events per second that can be routed between the nodes may be difficult to achieve. For state-of-the-art performance 1 GEv/s is required; for comparison a recent FPGA implementation of neural networks reaches on the order of 10 MEv/s [NL14]. I believe that it is possible to reach the required 1 GEv/s using a parallel routing system. Notably such a router can then be used to support FPGA implementations of other event-based algorithms presented here. The important result here from my perspective is that I give the key specifications that a massively parallel, event-based platform needs to meet in order to perform at the level of a standard architecture.

The PBMnet analysis (chapter 3) further forms the backbone of a theoretical explanation of the putative functional role of gamma oscillations [MMI15b] as underlying approximate inference in the brain. In appendix B I further provide an application of this theory to a perceptual multi-stability phenomenon that yields a testable prediction about the relationship between the coherence of gamma oscillations in nearby locations and the distribution of perceptual switching times. This relationship has not been investigated experimentally so far (to my knowledge) so that in the very least this suggests a straightforward method to rule out a putative functional role of gamma oscillations.

# Appendices

Appendix A

# Finite Size, Mismatched Winner-Take-All Networks

## A.1 Introduction

Because the WTA is a powerful computational primitive (see chapter 1) and because the typical units used in current iterations of neuromorphic hardware of our group are spiking neurons [QMCOSSI15], I investigate the link between the two. More specifically in this chapter I study in simulations how well a WTA functionality can be achieved by mismatched, finite-size spiking neurons.

In his PhD thesis Emre Neftci [Nef10] shows that practically surprisingly few spiking neurons emulated on a neuromorphic chip behave close to their mean field limit when configured into WTA circuits. I am quantifying and explaining this result in some more detail here.

Practically I did not make further use of the observations in this chapter (therefore I added it as an appendix only). There are two reasons for this. First of all section 3.10 describes a more direct method of using single spiking neurons in some of the contexts I study; secondly digital nodes, as the ones we ended up using for [MMI15a] do not suffer from the non-idealities I study here.

## A.2 Background

The mean-field theory in which WTA circuits composed of spiking neurons are often analysed [Nef10], assumes that there are very many neurons available for each state that is represented. Whether the brain operates using principles that make this requirement is not clear; there are many neurons available, but requiring large numbers of neurons to represent one state is inefficient (and a system that does not work at all if it is small, does not make sense in an evolutionary context). Clearly however, when building a neuromorphic

system, we would like to get as much computation or as many states as possible out of each neuron. The question I will therefore address in this section is: Does the mean-field approximation make useful predictions for finite-size systems?

Using many neurons has in the context of neuromorphic engineering an advantage besides the mathematical trickery of taking the continuum's limit. Since analog silicon neurons are mismatched, using them in groups larger than one is an effective way to 'average out' their differences, so that the groups are better matched than the single units would be. I therefore also tried to find out whether such finite-size systems behave very differently when their components are mismatched to different degrees.

Looking at mismatch and system-size simultaneously also allows me to evaluate where the greater room for improvement lies: In making larger systems or in reducing the mismatch of the components. But what do I mean with 'improvement'? The easiest way to be able to determine which of two systems is better, is to have a benchmark (a scalar that measures the system's performance). The benchmark I selected is a feature of the WTA functionality: How reliably can the system identify which of two inputs is greater, and amplify it while suppressing other inputs?

## A.3 Experimental Setup

**Neuron Model** The neuron model I chose closely matches the behaviour of the neurons on the ROLLS chip [QMCOSSI15]. The differential equations are that were integrated with a forward Euler method are:

$$\frac{C_{Mem}}{M \cdot I_\tau}\frac{dI_{mem}}{dt} = \frac{I_{mem}}{I_\tau} + \frac{\sum_i I_{in,i}}{1 + I_{mem}/I_{gth}} - I_\tau - I_{\mathrm{AHP}} + I_{\mathrm{FB}} \tag{A.1}$$

$$\frac{C_{\mathrm{AHP}}}{M \cdot I_\tau}\frac{dI_{\mathrm{AHP}}}{dt} = -I_{\mathrm{AHP}}. \tag{A.2}$$

When $I_{mem}$ exceeds a value $I_{mem,T}$ the neuron spikes, $I_{mem}$ gets reset to $I_{mem,R}$, $I_{FB}$ to $I_0$ and $I_{AHP}$ is increased by $I_{\mathrm{AHP,MAX}}$. Further we have

$$I_{FB} = \frac{I_0^{\frac{1}{u+1}} I_{mem}^{\frac{u}{u+1}}}{1 + \exp(-A(I_{mem} - I_{gth})} \tag{A.3}$$

and $I_0$ is the minimal current flowing through any transistor, $I_{mem}$ is the membrane current, $I_{\mathrm{AHP}}$ is the adaptation current, $I_{in,i}$ are the various input currents, other variables named $I_{\cdot}$ are bias currents. $A$ and $u$ are fitting parameters and $M = \frac{\kappa}{u_T}$ is a physical constant (at constant temperature).

These neurons communicate to each other through synapses that follow

$$\frac{C_{syn}}{M \cdot I_{\tau,s}} \frac{dI_E}{dt} = -I_E + \sum_i w_i \delta(t_i) \tag{A.4}$$

$\delta(t_i)$ is one whenever a spike arrives at the synapse, and zero otherwise. The equation for inhibitory synapses looks the same, but with a shorter time constant (10% of the excitatory one).

In the following simulations, the synaptic time constant $I_{\tau,S}$ and the threshold current $I_{gth}$ are subject to mismatch; these are the parameters whose mismatch I found to have the largest impact on the transfer function of the neuron, when subject to mismatch of a fixed relative size.

**Network Architecture** The neurons were grouped into populations of excitatory and inhibitory neurons, which were then connected according to the standard WTA connectivity: Two excitatory populations, each recurrently connected to itself, excite a shared inhibitory population that inhibits both excitatory populations equally strongly. The weights of the connections were chosen so that the system exhibits three stable states: No activity in either excitatory population, activity $X > 0$ in excitatory population 1 or activity $X > 0$ in population 2. Each excitatory populations receive inputs from its own large group (n=50) of Poisson neurons, called I1 and I2. Each neuron in I1 and I2 produces a Poisson spike train of average rate $32Hz$ or $40Hz$ for a short interval. $40Hz$ is termed the high input, $32Hz$ the low input. The full network is illustrated in figure A.1

The mean field system[1] will respond to this input in a simple way: The population that receives the higher input (either E1 or E2) will become active and suppress activity in the other population. When the input is removed, that population will remain active and the system will be in one of its stable states. The identity of the stable state that the system has reached is in mean field fully determined by the identity of the population that received the higher input. I could thus describe the operation this system performs as receiving a one bit input and storing it. (Note: there is a vanishingly small probability that the Poisson inputs will misrepresent the 'correct' state during the time it takes the system to reach its stable state; we will treat this probability as negligible.)

The question I now investigate is, how reliably a finite-size, mismatched system can perform this basic task of reading and storing a bit that is presented as two input Poisson rates. Two kinds of non-idealities now occur during this operation: Due to the finite size the population rates are not represented smoothly in time and due to mismatch not all neurons will respond the same way. I regard a successful trail as one in which the final

[1] To my knowledge it is not possible to derive analytically the mean-field behaviour of these equations. However e.g. constant-leak I&F neurons would behave like this and in simulation this seems to be true for these neurons too in the large system limit.

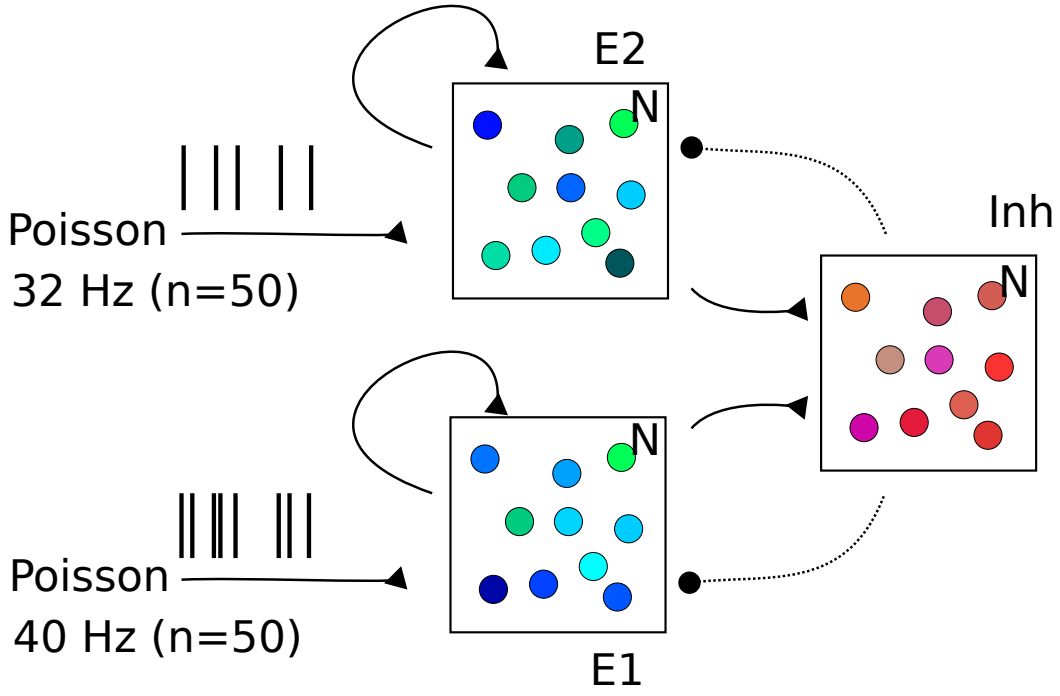



**Fig. A.1:** *The architecture of the simulated WTA network. Solid connections terminated by triangles indicate excitatory connections, dotted ones terminated by circles indicate inhibitory connections. Blue-green neurons are excitatory, red-purple ones inhibitory. Each 'box' indicates a neural population; these populations are made up of mismatched spiking neurons (coloured circles).*

active state of the WTA reflected correctly which of the input populations gave a higher input.

## A.4 Results

Figures A.2 and A.3 summarize the results of my simulations. The key observation is that a small amount of mismatch greatly improves the performance of the network. An intuitive explanation of this can be given as follows: In a zero mismatch scenario, all neurons spike at the same time giving rise to a maximal impact of the discretization introduced by the spiking mechanism; the neurons all resolve exactly the same input magnitudes with their effective threshold. At a slightly higher mismatch, effective thresholds differ and the neurons can spread out their spikes in time to more faithfully represent the input signal.

## A.5 Discussion

The fact that slightly mismatched spiking neurons are better at implementing WTA is interesting as it sheds new light on the facility with which even small groups of spiking analog VLSI neurons have been modelled by their mean-field description [GCMDBG12; Nef10].

A possible interpretation of the beneficial impact of mismatch is as a kind of stochastic resonance effect: Instead of injecting a single thresholded unit with noise to allow it to cross the threshold when an incoming signal is high, multiple units with mismatched (effective) thresholds can represent an incoming signal at better resolution that the single unit. Mismatch induces a so-to-speak 'spatial' analogon of the 'temporal' stochastic resonance effect.

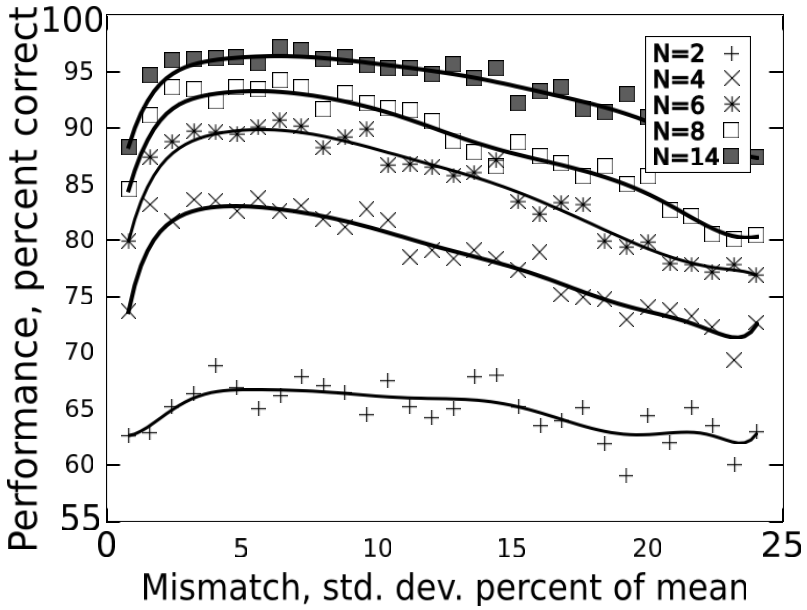



**Fig. A.2:** *Performance of networks (of various sizes) as a function of mismatch. Characteristic seem to be an initial sharp rise, then a slow decline.*

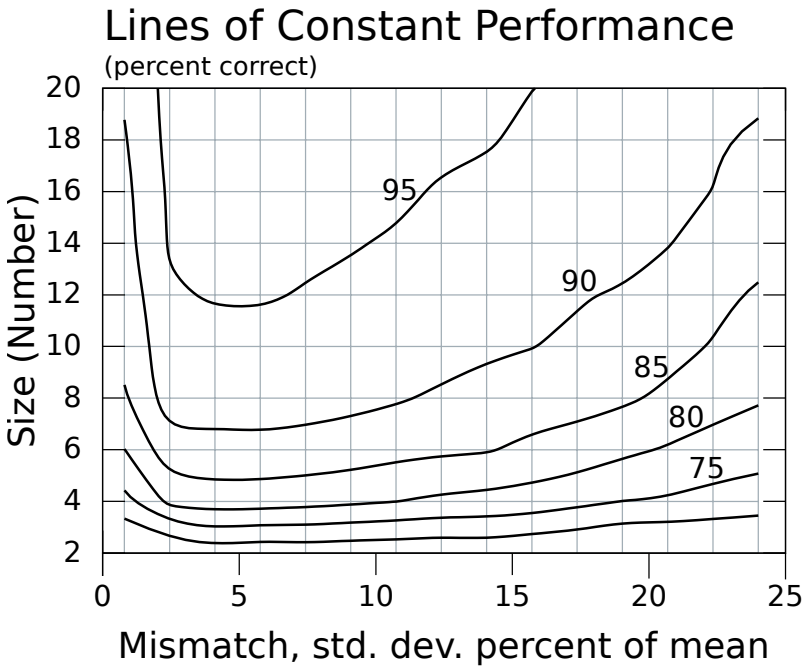



**Fig. A.3:** *Lines of constant performance in mismatch-size space. Clearly a small amount of mismatch is very useful for this task.*

Appendix B

# Perceptual Multistability in Coupled oWTA

This chapter, with the exception of the last two sections, is based on the paper "Rhythmic inhibition allows neural networks to search for maximally consistent states" by Hesham Mostafa, Lorenz K. Müller and Giacomo Indiveri [MMI15b].

## B.1 Introduction

In this chapter a biologically plausible learning rule, akin to the BCM rule, is introduced for learning in (single layer) oWTA networks. I show that an oWTA network can be used to model the psychophysical phenomenon of perceptual multistability, in which an ambiguous sensory input is interpreted in multiple self-consistent ways in sequence, and that an oWTA network can learn the connectivity required for this task from samples.

## B.2 A Plasticity Rule for Coupled oWTA

Learning from samples the probability distribution that an oWTA network or PBMnet should encode is a difficult problem that I addressed computationally in some detail in chapter 4. In this section I will study how a biologically plausible learning rule can function in an oWTA net.

A central question in a biologically plausible learning scheme is how the plasticity rule can distinguish between configurations that are input-imposed, and thus should be learned, and configurations that arise naturally when the network is running freely. One possible solution to this problem is to have the input impose a particular configuration for a long time and thus distinguish this configuration from the others by virtue of its longer persistence. This, however, leads to slow learning and there is no guarantee that a configuration generated in the free-running network will not persist just as long. This central question

also arises in stochastic connectionist architectures that represent a probability distribution by sampling and that learn the probability distribution by example such as restricted Boltzmann machines (RBM). The learning rule used in RBMs [AHS85; Hin02] assumes the network has access to a signal indicating whether a configuration is input-imposed or not. This signal switches the plasticity rule modulating the weights in the network between Hebbian and anti-Hebbian modes. It is not clear what the biological analogue of such a scheme could be.

To distinguish an input-imposed configuration as a configuration that should be learned, i.e, that should serve as a model for consistent configurations, external input synchronizes the activity of the WTA circuits it targets so that the oscillatory inhibition in these WTA circuits has a common frequency and a zero phase difference.[1] This way, the full or partial configurations imposed by the input have the distinguishing dynamical feature of synchronized WTA circuits. Synchronization is thus used as a dynamical marker for configurations that should be learned. How the input is able to synchronize the WTA circuits, is not explicitly modelled; the synchronization is imposed by a periodic external signal to which the local oscillators are coupled.

The following plasticity rule acts on the weights of the inter-WTA coupling connections.

$$\begin{aligned} \frac{\partial w(t)}{\partial t} = &+ d(w) \\ &+ \eta_+(r_{pre},r_{post}) \cdot [r_{pre}(t) - \theta] \cdot [r_{post}(t) - \theta] \\ &- \eta_-(r_{pre},r_{post}) \cdot [r_{pre}(t) - \theta] \cdot [\theta - r_{post}(t)] \end{aligned} \tag{B.1}$$

where

$$d(w) = \begin{cases} d_{up} & \text{if } w(t) > w_{mid} \\ -d_{down} & \text{otherwise} \end{cases}$$

$$\eta_+(r_{pre},r_{post}) = \begin{cases} \eta_{up} & \text{if } r_{pre}(t) > \theta \text{ and } r_{post}(t) > \theta \\ 0 & \text{otherwise} \end{cases}$$

$$\eta_-(r_{pre},r_{post}) = \begin{cases} \eta_{down} & \text{if } r_{pre}(t) > \theta \text{ and } r_{post}(t) \leqslant \theta \\ 0 & \text{otherwise} \end{cases}$$

$w(t)$ is the weight. $r_{pre}(t)$ and $r_{post}(t)$ are the firing rates of the source (presynaptic) and target (postsynaptic) populations respectively. This rule is a variation of the Bienenstock-Cooper-Munro (BCM) rule [BCM82] with hard weight bounds $w_{min}$ and $w_{max}$ (not shown in the equation) and the requirement that $r_{pre}(t)$ has to exceed a threshold, $\theta$, in order to induce any change in the weight $w(t)$. If this requirement is met, potentiation is induced if the postsynaptic activity, $r_{post}(t)$, is above the threshold $\theta$, and depression is induced if

[1] The idea of using synchronicity as a dynamical marker to distinguish learning and sampling phases in oWTA networks is due to Hesham Mostafa.

the postsynaptic activity is below the threshold. The rates of potentiation and depression induction are controlled by $\eta_{up}$ and $\eta_{down}$ respectively. The rule captures the way potentiation and depression induction depend on the pre- and post-synaptic firing rates [STN01]. The rule contains a second component that slowly forces the connection weight to either $w_{min}$ or $w_{max}$ depending on whether the weight is below or above $w_{mid} = \frac{1}{2}(w_{max} + w_{min})$ respectively. The rule is thus bistable. The strength of the bistability drift is controlled by $d_{up}$ and $d_{down}$. Bistable plasticity is computationally less powerful than its counterpart with continuous stable weights [AF92; Som87], but can be argued to be more biologically realistic, due to noise-tolerance and finite synaptic information content.

The plasticity rule in Eq. B.1 results in the functional effect of enhanced plasticity in the inter-WTA connections when the WTA circuits are synchronized. In order for a depressed connection to potentiate, both pre- and postsynaptic rates need to be large at the same time for many consecutive cycles in order to overcome the bistability drift. This will only reliably happen if the oscillatory inhibition in the pre- and post-synaptic WTA circuits have a phase difference that is around zero for many cycles so that the peaks of excitatory activity in the WTA circuits coincide.

## B.3 Perceptual Multistability in oWTA

Binocular rivalry is a form of perceptual multi-stability that has been particularly well studied, both in experiments [MG05] as well as in theory [GVT12]. A subject is shown two differently oriented gratings (90 degrees between their orientations, i.e, maximally different), one to each eye. The reaction to such a mismatched stimulus is not that subjects perceive a blending of the two, but rather that their interpretation of the stimulus switches between the two orientations [MG05]. The subjects' interpretation of these inputs fits well into the "perception as inference" framework: due to their experience the subjects have a strong prior belief that their eyes should deliver consistent input; two greatly mismatched inputs are treated as contradictory evidence for two distinct possible underlying states of the world (this is the "reasonable" interpretation when one object occludes another). In particular the switching between the two states indicates that the probabilities in this task are represented by samples of a distribution, rather than as full distributions at any time point. oWTA networks can learn what constitutes a consistent input using the biologically plausible learning rule from the previous section and then reproduce perceptual multistability dynamics when faced with inconsistent inputs.

### B.3.1 Network Model

The network is composed of $n_{hid} = 6$ hidden and $n_{in} = 6$ input WTA circuits that each undergo oscillatory inhibition. Each WTA has six competing excitatory populations, $n_c = 6$, and can thus take one of six different states (corresponding to six different orientations).

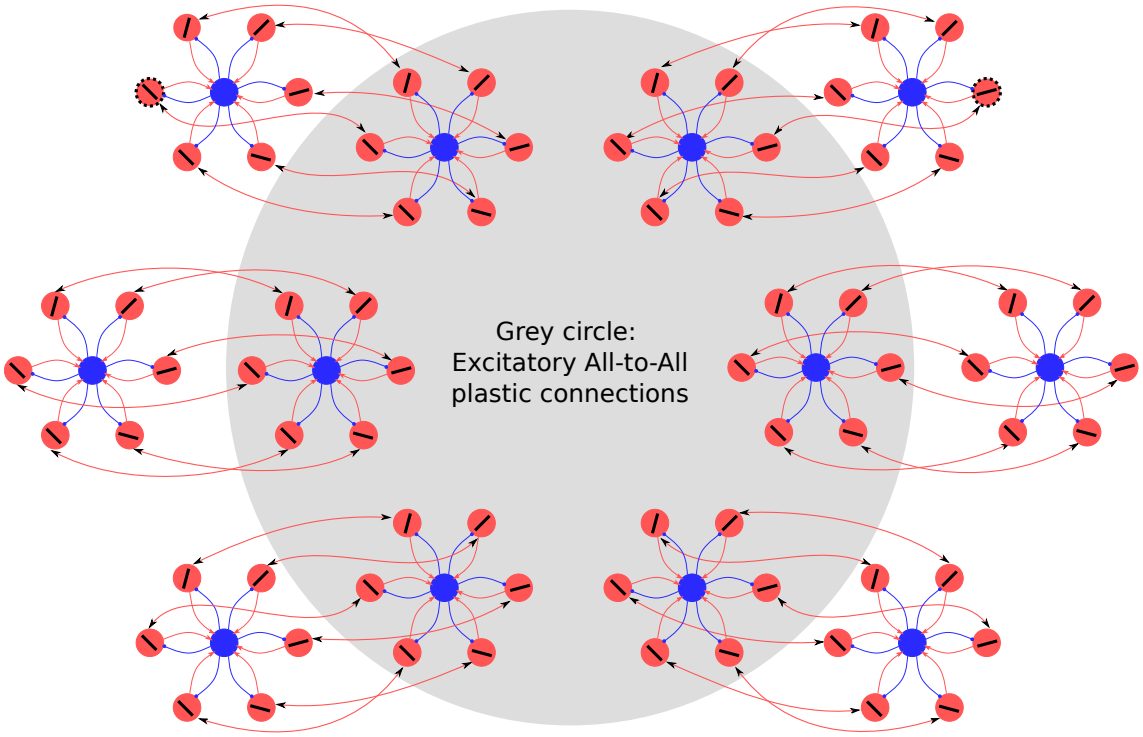



**Fig. B.1:** *The connectivity of the network performing the perceptual multi-stability task. The hidden WTA circuits are inside the grey circle while the input WTA circuits are outside. All explicitly drawn connections are fixed connections. The six hidden WTA circuits have all-to-all plastic connectivity, i.e, each excitatory population in one hidden WTA connects to each excitatory population in the five other hidden WTA circuits. Each WTA has six excitatory populations and each excitatory population codes for a* $30^\circ$ *range of orientations centered on the bar orientation drawn on the excitatory population. Each WTA receives local oscillatory inhibition (not shown). The recurrent excitatory connections are also not shown. The input WTA circuits can get external input that biases the winner selection towards a certain population (dashed outlines for the example in Fig. B.2c).*

There is no conceptual problem in using WTA circuits having more states (more excitatory populations in the WTA) or using WTA circuits with unequal number of states. Smaller number of states per WTA in the current context corresponds to broader population tuning curves for the excitatory populations. $n_c = 6$ implies that each excitatory population codes for a range of orientations of $\frac{180^\circ}{6} = 30^\circ$.

The full network connectivity is illustrated in Fig. B.1. Input and hidden WTA circuits are set up in pairs. Input WTA circuits get external input while hidden WTA circuits do not. Each input WTA connects to one hidden WTA so that populations of similar orientations are bidirectionally coupled. The connectivity between hidden WTA circuits is all-to-all and initialized using random weights: $w^i_{init} \in [w_{max}, w_{min}]$ that are *plastic* and follow the plasticity rule in Eq. B.1.

The activity of the hidden WTA circuits encodes an angle which represents the current guess of the network at the true orientation which is communicated to it by the input

WTA circuits. To decode the network activity, first scale the angles of the normalized preferred directions of the excitatory populations from the $0-180°$ range to the $0-360°$ range. We then perform a vector addition of the modified preferred directions of all active populations of the hidden WTA circuits to obtain the two quantities:

$$\theta = \frac{1}{2} atan2\left(\sum_i \delta_i y_i, \sum_i \delta_i x_i\right) \tag{B.2}$$

$$r = \sqrt{\left(\sum_i \delta_i y_i\right)^2 + \left(\sum_i \delta_i x_i\right)^2} \tag{B.3}$$

where $atan2$ is the two argument quadrant adjusted inverse of the tangent, $x_i$ ($y_i$) is the x-coordinate (y-coordinate) of the modified preferred direction of population $i$ and $\delta_i \in \{0,1\}$ is the indicator of the activity of population $i$. The summation runs over all excitatory populations in the hidden WTA circuits. The factor $\frac{1}{2}$ in Eq. B.2 puts the decoded angle back in the $0-180°$ range. $r$, the magnitude of the decoded activity vector, can be understood as the confidence of the system in its current angle estimate. When all hidden WTA circuits encode a different angle, $r$ takes a value close to zero, when they all encode the same angle, it takes the maximal value $n_{hid} = 6$.

The input WTA circuits receive different inputs at different times that bias the network to preferentially visit certain configurations. How exactly the input to the input WTA circuits affects the distribution of visited configurations depends on the connectivity between the hidden WTA circuits; in Bayesian terms, the posterior depends on both external evidence and the prior.

### B.3.2 Training and Testing

The simulations I run have two distinct phases, a training phase during which the system learns priors (the strength of synaptic weights connecting the hidden WTA circuits) from examples that are presented to the input WTA circuits, and a testing phase during which I present some various inputs and decode the activity of the hidden WTA circuits. During the two phases I use *the same parameters*; learning is enabled/disabled by providing/withholding a synchronizing oscillatory input to the inhibitory oscillators of all input and hidden WTA circuits.

"Training" the network refers to the following procedure: For 30 seconds, I provide input that clamps the states of all input WTA circuits so that they are all encoding the same orientation, while enabling the global synchronizing oscillation. Within these 30 seconds, the input cycles through the $n_c = 6$ directions each WTA can encode. The configuration of the input WTA circuits is thus forced to represent an unambiguous orientation which would act to influence the hidden WTA circuits so that they are also encoding the same

unambiguous orientation. The connections between the hidden WTA circuits change to represent the prior that all hidden WTA circuits usually encode the same orientation.

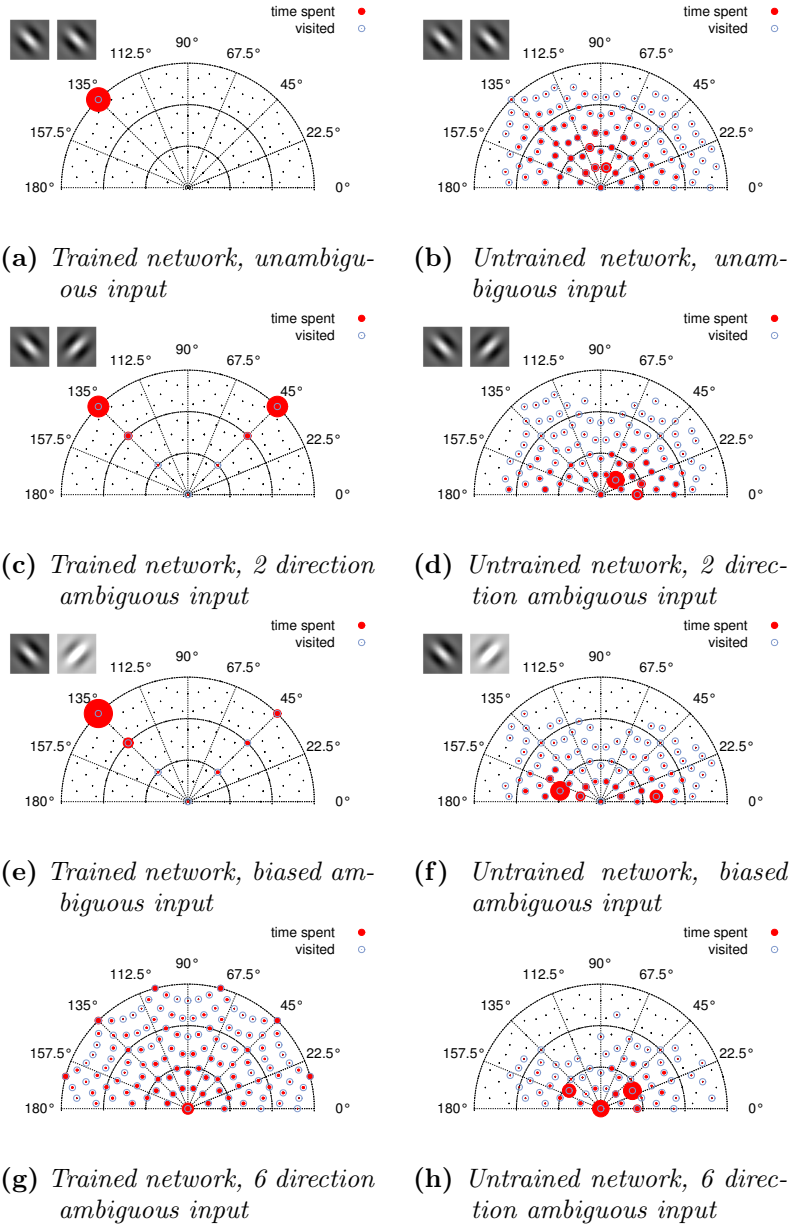
During testing, I provide inputs that clamp the states of a subset, or of all, input WTA circuits to certain orientations for 4 minutes while withholding the global synchronizing oscillation. I record the activity of the hidden WTA circuits and decode it into an angle and a magnitude as outlined in Eqs B.2 and B.3. I interpret the normalized time histogram of this decoded signal as the probability the network assigns to a given $r$, $\theta$ pair.

The first test stimulus I consider is an unambiguous input: one input WTA receives input that clamps it at one orientation. For the trained network there is a consistent state corresponding to this input (all WTA circuits encode that direction) and the network goes to that solution as can be seen in Fig. B.2a. In contrast the untrained network has random weights in the connections coupling the hidden WTA circuits. There is no consistent solution and it explores whatever probability distribution is encoded in its weights with some mild bias from the input, see Fig. B.2b. Secondly, I consider an ambiguous input corresponding to the presentation of two oppositely oriented gratings to two eyes: one input WTA is clamped at 45° while another input WTA is clamped at 135°. As Fig. B.2c shows, the trained network flips between these two orientations and spends the same amount of time in each. The trained network can thus reproduce the two key observations of multi-stable perception. The untrained network in contrast cannot make sense of this input and produces, like in the previous case, a distribution dominated by its random connectivity.

As a test of the soundness of the inference operation, I further provided an ambiguous input with unequal strength (two instead of one input WTA receive input at 135°) while only one receives input at 45°. A corresponding experiment would be to display two gratings with different contrasts. The trained system performs the correct inference and samples the 135° state more frequently, see Fig. B.2e. Finally I provided an input that does not correspond to any binocular rivalry experiment in which each input WTA receives input that clamps it to different direction from the other input WTA circuits. The trained network resolves this contradictory input by visiting all trained states equally often, but also spends some time in very low confidence states (like the completely "agnostic" state in the center).

To be able to make a quantitative comparison to human experiments, I investigate the perceptual switching times the model produces when presented with an ambiguous stimulus. I add an 'integrative' component to the system based on [WHSW07] in which a bistable decision task for random dot stimuli is modelled: A bistable attractor network receives time-varying inputs and settles into one attractor state. The time varying input in this case is produced by the hidden WTAs and the attractor network slowly integrates their outputs. The bistable attractor's population with the higher activity corresponds to the current percept, see Fig. B.3a.

The switching times I obtain closely match human experimental data (from [MG05]),




**(a)** *Trained network, unambiguous input*

**(b)** *Untrained network, unambiguous input*

**(c)** *Trained network, 2 direction ambiguous input*

**(d)** *Untrained network, 2 direction ambiguous input*

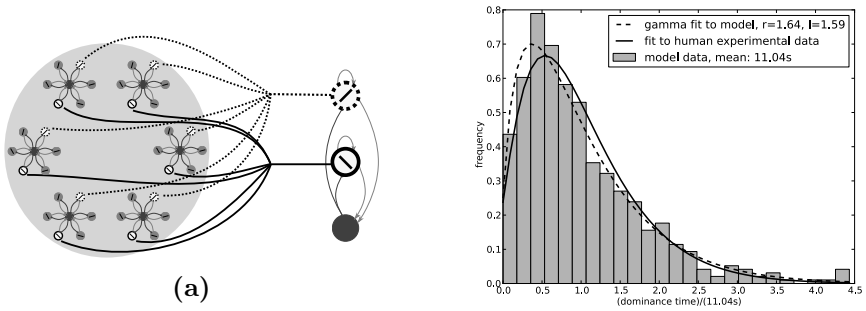
**(e)** *Trained network, biased ambiguous input*

**(f)** *Untrained network, biased ambiguous input*

**(g)** *Trained network, 6 direction ambiguous input*

**(h)** *Untrained network, 6 direction ambiguous input*

**Fig. B.2:** *The "probability distribution" of the orientation vector decoded from the network in Fig. B.1 under different conditions. Black dots indicate states that can be encoded by the network, and blue empty circles indicate states that were visited by the network during the simulation. The area of the filled red circle at each visited location is proportional to the time spent at (the probability of) that state (scale differs between plots). The inset Gabor patches indicate the stimuli presented to the left and right eye in analogous experimental settings. (a) A network trained on consistent inputs propagates the angle input to one WTA to the other WTA circuits and stably represents that angle. (b, d, f, h) An untrained network spends most of the time in "low confidence states" where the hidden WTA circuits encode different angles. (c) The trained network interprets input at 45° and 135° as either of the two. (e) Increasing the number of input WTA circuits encoding 135° to two makes that interpretation more probable compared to the 45° interpretation (g) The trained network under fully conflicting input preferentially visits consistent states, where all populations represent the same direction. "Low confidence" states are also visited.*




**(a)** **(b)**

**Fig. B.3:** *(a)Readout WTA performing the perceptual decision similar to [WHSW07]. (b)The perceptual switching times produced by our model follow a gamma distribution and agree with experimental data from [MG05].*

see Fig. B.3b. The histogram is approximately a gamma-distribution. The mean time between switches is around 11 seconds, with a maximum of 49 seconds. In this model the gamma distribution arises from the interplay between the 'integrative' readout and the hidden WTAs: The hidden WTA switch between interpretations at intervals distributed approximately according to an exponential distribution (this gives rise to the tail of the distribution produced by the full system); the 'integrative' attractor has a certain inertia caused by the size of the integration window, which makes very fast switches much harder (this gives rise to the slow rise of the full system's distribution).

Besides fitting experimental data, this model also makes an experimentally verifiable prediction: Greater variability in the localized oscillators leads to a more exploratory behaviour of the system and thus to on average shorter perceptual switching times. The biological correlate of the localized oscillators are gamma oscillations that have long been speculated to play a role in high-level cognitive processing (e.g. [MM+95]).

## B.4 Perceptual Multi-Stability on Neuromorphic Hardware

I attempted to recreate the model of the previous section in neuromorphic hardware, specifically ROLLS [QMCOSSI15]. Due to limitations in the number of free parameters, maximal input spike-rates and the maximal current 'virtual' synapses can supply per spike, it seems impossible to me to simultaneously set up a WTA with the right properties and to periodically shut it down at gamma frequency through external or internal inhibition.

It is possible to modify the model to rely on a different mechanism of attractor destabilization: Finite-size spiking effects. A WTA implemented by few spiking units can be intrinsically unstable; if all neurons happen to spike close together the resulting strong inhibition can destroy the attractor. However this fails to capture the key mechanism (inhibitory oscillations) that was the focus of this investigation.

On the other hand it is interesting to see that a different mechanism of attractor destabilization leads to similar results. An initial study of this was done by my colleague Federico Corradi in [CHGI15]. A marked difference to the framework used here is that the composability of the attractors is not straight-forward. The probabilistic switching relies on Poisson inputs, while the output is non-Poisson, but highly coherent on short time-scales.

How the finite-size spiking destabilization could be used in a composable system is, as far as I know, not well studied. In a brief investigation I found that mismatch in combination with finite-size seem to be sufficient to obtain fairly irregular output responses from competing WTA modes even to regular input, though not with as low auto-correlations as with Poisson inputs.

## B.5 Discussion

The key outcome of this perceptual multi-stability model, is that it yields a testable prediction about the oWTA model of gamma oscillations: If the local gamma oscillators have higher coherence, it is less likely for a minority input to affect the overall system state and consequently switching times become longer and the bias towards states with stronger input becomes stronger. To my knowledge a link between gamma coherence and perceptual switching times has not been experimentally studied.

A confirmation of this link would be a significant result, because it would indicate a high level functional role of gamma rhythms. From this point it would be interesting to study correlations of gamma coherence with high-level brain functions and potentially methods of affecting gamma coherence.

# Bibliography


[ABR64] A Aizerman, Emmanuel M Braverman, and LI Rozoner. "Theoretical foundations of the potential function method in pattern recognition learning". In: *Automation and remote control* 25 (1964), pp. 821–837.

[AF92] D.J. Amit and S. Fusi. "Constraints on learning in dynamic synapses". In: *Network: Computation in Neural Systems* 3.4 (1992), pp. 443–464.

[AF94] D.J. Amit and S. Fusi. "Dynamic learning in neural networks with material synapses". In: *Neural Computation* 6 (1994), p. 957.

[AHS85] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. "A learning algorithm for Boltzmann machines". In: *Cognitive science* 9.1 (1985), pp. 147–169.

[ALCRTW11] Matthew Ainsworth, Shane Lee, M.O. Cunningham, A.K. Roopun, R.D. Traub, and N.J. Kopelland M.A. Whittington. "Dual gamma rhythm generators control interlaminar synchrony in auditory cortex". In: *The Journal of Neuroscience* 31.47 (2011), pp. 17040–17051.

[BBBLPDTWFB10] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. "Theano: A CPU and GPU math compiler in Python". In: *Proceedings of the Python for Scientific Computing Conference, SciPy*. 2010.

[BBNM11] Lars Buesing, Johannes Bill, Bernhard Nessler, and Wolfgang Maass. "Neural dynamics as sampling: A model for stochastic computation in recurrent networks of spiking neurons". In: *PLoS computational biology* 7.11 (2011), e1002211.

[BCM82] E.L. Bienenstock, L.N. Cooper, and P.W. Munro. "Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex". In: *Jour. Neurosci.* 2.1 (1982), pp. 32–48. eprint: `http://www.jneurosci.org/cgi/reprint/2/1/32.pdf`.

[BD04] György Buzsáki and Andreas Draguhn. "Neuronal oscillations in cortical networks". In: *Science* 304.5679 (2004), pp. 1926–1929.

[BDHJ] Anton Belov, Daniel Diepol, Marjin Heule, and Matti Järvisalo.

[BG05] R. Brette and W. Gerstner. "Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity". In: *Journal of Neurophysiology* 94 (2005), pp. 3637–3642.

[BGMCCBAIAMB14] Ben Varkey Benjamin, Peiran Gao, Emmett McQuinn, Swadesh Choudhary, Anand R Chandrasekaran, J Bussat, R Alvarez-Icaza, JV Arthur, PA Merolla, and K Boahen. "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations". In: *Proceedings of the IEEE* 102.5 (2014), pp. 699–716.

[Bis06] C.M. Bishop. *Pattern recognition and machine learning*. Springer New York, 2006.

[BL14] Johannes Bill and Robert Legenstein. "A compound memristive synapse model for statistical learning through STDP in spiking neural networks". In: *Frontiers in Neuroscience* 8 (2014), p. 412.

[BR08] Adam B Barrett and Mark CW van Rossum. "Optimal learning rules for discrete synapses". In: *PLoS computational biology* 4.11 (2008), e1000230.

[Bra] *Brain-inspired multiscale computation in neuromorphic hybrid systems (BrainScaleS)*. FP7 269921 EU Grant. 2011–2015.

[BS12] Adrian Balint and Uwe Schöning. "Choosing probability distributions for stochastic local search and the role of make versus break". In: *Theory and Applications of Satisfiability Testing–SAT 2012*. Springer, 2012, pp. 16–29.

[BSF07] J. Brader, W. Senn, and S. Fusi. "Learning real world stimuli in a neural network with spike-driven synaptic dynamics". In: *Neural Computation* 19 (2007), pp. 2881–2912.

[BSXSS10] S.P. Burns, P. Samuel, D. Xing, M.J. Shelley, and R.M. Shapley. "Searching for autocoherence in the cortical network with a time-frequency analysis of the local field potential". In: *The Journal of Neuroscience* 30.11 (2010), pp. 4033–4047.

[BW12] György Buzsáki and Xiao-Jing Wang. "Mechanisms of gamma oscillations". In: *Annual review of neuroscience* 35 (2012), pp. 203–225.

[BZ10] Gerhard Bohm and Günter Zech. *Introduction to statistics and data analysis for physicists*. DESY, 2010.

[Caf98] Russel E Caflisch. "Monte carlo and quasi-monte carlo methods". In: *Acta numerica* 7 (1998), pp. 1–49.

[CBD14] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. "Low precision arithmetic for deep learning". In: *arXiv preprint arXiv:1412.7024* (2014).

[CBDDSCFD03] E. Chicca, D. Badoni, V. Dante, M. D'Andreagiovanni, G. Salina, L. Carota, S. Fusi, and P. Del Giudice. "A VLSI recurrent network of integrate–and–fire neurons connected by plastic synapses with long term memory". In: *IEEE Transactions on Neural Networks* 14.5 (2003), pp. 1297–1307. DOI: 10.1109/TNN.2003.816367.

[CCMKZDTM09] J.A. Cardin, Marie Carlén, Konstantinos Meletis, Ulf Knoblich, Feng Zhang, Karl Deisseroth, Li-Huei Tsai, and C.I. Moore. "Driving fast-spiking cells induces gamma rhythm and controls sensory responses". In: *Nature* 459.7247 (2009), pp. 663–667.

[CHGI15] F. Corradi, You H., M. Giulioni, and G. Indiveri. "Decision Making and Perceptual Bistability in Spike-Based Neuromorphic VLSI Systems". In: *International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2015.

[CL11] Chih-Chung Chang and Chih-Jen Lin. "LIBSVM: a library for support vector machines". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 2.3 (2011), p. 27.

[CTB07] Gregor Cijan, Tadej Tuma, and A Burmen. "Modeling and simulation of MOS transistor mismatch". In: *Proc. 6th EUROSIM* (2007), pp. 1–8.

[DJ99] Włodzisław Duch and Norbert Jankowski. "Survey of neural transfer functions". In: *Neural Computing Surveys* 2.1 (1999), pp. 163–212.

[DM04] R.J. Douglas and K.A.C. Martin. "Neural Circuits of the Neocortex". In: *Annual Review of Neuroscience* 27 (2004), pp. 419–51.

[DM07] R.J. Douglas and K. Martin. "Recurrent neuronal circuits in the neocortex". In: *Current Biology* 17.13 (2007), R496–R500.

[DNBCLP] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. “Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing”. In: ().

[FBOL10] József Fiser, Pietro Berkes, Gergő Orbán, and Máté Lengyel. “Statistically optimal perception and learning: from behavior to neural representations”. In: *Trends in cognitive sciences* 14.3 (2010), pp. 119–130.

[FDA05] S. Fusi, P.J. Drew, and L.F. Abbott. “Cascade models of synaptically stored memories”. In: *Neuron* 45 (2005), pp. 599–611.

[FGTP14] S.B. Furber, F. Galluppi, S. Temple, and L.A. Plana. “The SpiNNaker Project”. In: *Proceedings of the IEEE* 102.5 (2014), pp. 652–665. ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2304638.

[Fri09] Pascal Fries. “Neuronal gamma-band synchronization as a fundamental process in cortical computation”. In: *Annual review of neuroscience* 32 (2009), pp. 209–224.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep sparse rectifier neural networks”. In: *International Conference on Artificial Intelligence and Statistics*. 2011, pp. 315–323.

[GBVRGPGD14] D Garbin, O Bichler, E Vianello, Q Rafhay, C Gamrat, L Perniola, G Ghibaudo, and B DeSalvo. “Variability-tolerant convolutional neural network for pattern recognition applications based on oxram synapses”. In: *Electron Devices Meeting (IEDM), 2014 IEEE International*. IEEE. 2014, pp. 28–4.

[GCMDBG12] M. Giulioni, P. Camilleri, M. Mattia, V. Dante, J. Braun, and P. Del Giudice. “Robust working memory in an asynchronously spiking neural network realized in neuromorphic VLSI”. In: *Frontiers in Neuroscience* 5.149 (2012). ISSN: 1662-453X. DOI: 10.3389/fnins.2011.00149.

[GG15] Yarin Gal and Zoubin Ghahramani. “Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning”. In: *arXiv preprint arXiv:1506.02142* (2015).

[GVT12] Samuel J Gershman, Edward Vul, and Joshua B Tenenbaum. “Multistability and perceptual inference”. In: *Neural computation* 24.1 (2012), pp. 1–24.

[GWFMCB13a] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. "Maxout networks". In: *arXiv preprint arXiv:1302.4389* (2013).

[GWFMCB13b] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. "Maxout networks". In: *arXiv preprint arXiv:1302.4389* (2013).

[Hal60] John H Halton. "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals". In: *Numerische Mathematik* 2.1 (1960), pp. 84–90.

[HC68] J. M. Hammersley and Peter Clifford. "Markov fields on finite graphs and lattices". In: (1968).

[Hin02] Geoffrey E Hinton. "Training products of experts by minimizing contrastive divergence". In: *Neural computation* 14.8 (2002), pp. 1771–1800.

[Hin10] Geoffrey Hinton. "A practical guide to training restricted Boltzmann machines". In: *Momentum* 9.1 (2010), p. 926.

[Hoc91] Sepp Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen". In: *Diploma, Technische Universität München* (1991).

[Hop82] J.J. Hopfield. "Neural networks and physical systems with emergent collective computational abilities". In: *Proceedings of the national academy of sciences* 79.8 (1982), pp. 2554–2558.

[HOT06] G.E. Hinton, S. Osindero, and Y.W. Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[HS04] Holger H Hoos and Thomas Stützle. *Stochastic local search: Foundations & applications*. Elsevier, 2004.

[HS25] H. von Helmholtz and J.P.C Southall. *Treatise on physiological optics. III. The perceptions of vision.* New York: Optical Society of America, 1925.

[HS83] Geoffrey E Hinton and Terrence J Sejnowski. "Optimal perceptual inference". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. IEEE Piscataway, NJ. 1983, pp. 448–453.

[HS98] Holger H Hoos and Thomas Stützle. "Satlib–the satisfiability library". In: *Web site at: http://www. satlib. org* (1998).

[HSKSS12] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R.R. Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors". Submitted on 03 Jul. 2012. 2012.

[HT85] J.J. Hopfield and D.W. Tank. ""Neural" computation of decisions in optimization problems". In: *Biological cybernetics* 52.3 (1985), pp. 141–152.

[Ind02] G. Indiveri. "Neuromorphic Bistable VLSI Synapses with Spike-Timing-Dependent Plasticity". In: *Advances in Neural Information Processing Systems*. Vol. 15. Cambridge, MA, USA: MIT Press, 2002, pp. 1091–1098.

[JLG04] R. Jolivet, T.J. Lewis, and W. Gerstner. "Generalized integrate-and-fire models of neuronal activity approximate spike trains of a detailed model to a high degree of accuracy". In: *Journal of neurophysiology* 92 (2004), pp. 959–976.

[KH09] Alex Krizhevsky and Geoffrey Hinton. "Learning multiple layers of features from tiny images". In: *Computer Science Department, University of Toronto, Tech. Rep* 1.4 (2009), p. 7.

[KM14] Hanna Kamyshanska and Roland Memisevic. "The potential energy of an autoencoder". In: (2014).

[LG13] Subhaneil Lahiri and Surya Ganguli. "A memory frontier for complex synapses". In: *Advances in Neural Information Processing Systems*. 2013, pp. 1034–1042.

[LM11] Hugo Larochelle and Iain Murray. "The neural autoregressive distribution estimator". In: *Journal of Machine Learning Research* 15 (2011), pp. 29–37.

[Maa00] W. Maass. "On the computational power of winner-take-all". In: *Neural Computation* 12.11 (2000), pp. 2519–2535.

[MAAICSAJIGN+14] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. "A million spiking-neuron integrated circuit with a scalable communication network and interface". In: *Science* 345.6197 (2014), pp. 668–673.

[Mac+67] James MacQueen et al. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. California, USA. 1967, pp. 281–297.

[Mac92] Alan K Mackworth. "Constraint satisfaction problems". In: *Encyclopedia of AI* 285 (1992), p. 293.

[MBWWDMF13] M.Rigotti, O. Barak, M.R. Warden, X.J. Wang, N.D. Daw, E.K. Miller, and S. Fusi. "The importance of mixed selectivity in complex cognitive tasks". In: *Nature* (2013). ISSN: 1476-4687. DOI: 10.1038/nature12160.

[Mea89] C.A. Mead. *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley, 1989.

[MG05] Pascal Mamassian and Ross Goutcher. "Temporal dynamics in bistable perception". In: *Journal of Vision* 5.4 (2005), p. 7.

[MI15] Lorenz K Muller and Giacomo Indiveri. "Rounding Methods for Neural Networks with Low Resolution Synaptic Weights". In: *arXiv preprint arXiv:1504.05767* (2015).

[Mil04] John Milnor. "On the concept of attractor". In: *The Theory of Chaotic Attractors*. Springer, 2004, pp. 243–264.

[MM+95] William A MacKay, Antonio J Mendoncc, et al. "Field potential oscillatory bursts in parietal cortex before and during reach". In: *Brain research* 704.2 (1995), pp. 167–174.

[MMI13] H. Mostafa, L. K. Müller, and G. Indiveri. "Recurrent networks of coupled Winner-Take-All oscillators for solving constraint satisfaction problems". In: *Advances in Neural Information Processing Systems (NIPS)*. Ed. by C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger. Vol. 26. 2013, pp. 719–727.

[MMI15a] Hesham Mostafa, L. K. Müller, and Giacomo Indiveri. "An event-based architecture for solving constraint satisfaction problems". In: *Nature Communications* (2015). (In press).

[MMI15b] Hesham Mostafa, L. K. Müller, and Giacomo Indiveri. "Rhythmic inhibition allows neural networks to search for maximally consistent states". In: *Neural computation* (2015). (in press).

[Mni] *The MNIST database of handwritten digits*. Yann LeCun's website. 2012.

[Mos11] Hesham Mostafa. Private Communication. 2011-2015.

[MP43] W.S. McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *Bull. Math. Biophys.* 5 (1943), pp. 115–133.

[MRPCAPW11] Gilberto Medeiros-Ribeiro, Frederick Perner, Richard Carter, Hisham Abdalla, Matthew D Pickett, and R Stanley Williams. "Lognormal switching times for titanium dioxide bipolar memristors: origin and resolution". In: *Nanotechnology* 22.9 (2011), p. 095702.

[MRRTT53] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. "Equation of state calculations by fast computing machines". In: *The journal of chemical physics* 21 (1953), p. 1087.

[MTRWGSW04] H. Markram, M. Toledo-Rodriguez, Y. Wang, A. Gupta, G. Silberberg, and C. Wu. "Interneurons of the neocortical inhibitory system". In: *Nature Reviews Neuroscience* 5.10 (2004), pp. 793–807.

[NDPKDC14] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs. "Event-Driven Contrastive Divergence for Spiking Neuromorphic Systems". In: *Frontiers in Neuroscience* 7.272 (2014). ISSN: 1662-453X. DOI: 10.3389/fnins.2013.00272.

[Nef10] E. Neftci. "Towards VLSI Spiking Neuron Assemblies as General-Purpose Processors". Ph.D. thesis. ETH Zürich, 2010.

[NL14] D. Neil and S.-C. Liu. "Minitaur, an Event-Driven FPGA-Based Spiking Network Accelerator". In: *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on* PP.99 (2014), pp. 1–1. DOI: 10.1109/TVLSI.2013.2294916.

[NPM09] B. Nessler, M. Pfeiffer, and W. Maass. "STDP enables spiking neurons to detect hidden causes of their inputs". In: *Advances in Neural Information Processing Systems (NIPS)*. Ed. by Y. Bengio, D. Schuurmans, J. Lafferty, C.K. I. Williams, and A. Culotta. Vol. 22. 2009, pp. 1357–1365.

[ONLDP13] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer. "Real-time classification and sensor fusion with a spiking deep belief network". In: *Frontiers in Neuroscience* 7.178 (2013). ISSN: 1662-453X. DOI: 10.3389/fnins.2013.00178.

[Pap91] C.H. Papadimitriou. "On selecting a satisfying truth assignment". In: *Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on*. IEEE. 1991, pp. 163–169.

[PBM11] Dejan Pecevski, Lars Buesing, and Wolfgang Maass. "Probabilistic inference in general graphical models through sampling in stochastic networks of spiking neurons". In: *PLoS computational biology* 7.12 (2011), e1002294.

[PCDGPTB11] L.A. Plana, D. Clark, S. Davidson, J. Garside, E. Painkras, S. Temple, and J. Bainbridge. "SpiNNaker: Design and Implementation of a GALS Multicore System-on-Chip". In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 7.4 (2011), p. 17.

[PWDFSERSM12] Robert Preissl, Theodore M Wong, Pallab Datta, Myron Flickner, Raghavendra Singh, Steven K Esser, William P Risk, Horst D Simon, and Dharmendra S Modha. "Compass: A scalable simulator for an architecture for cognitive computing". In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press. 2012, p. 54.

[QMCOSSI15] Ning Qiao, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumislawska, and Giacomo Indiveri. "A Reconfigurable On-line Learning Spiking Neuromorphic Processor comprising 256 neurons and 128K synapses". In: *Frontiers in Neuroscience* 9.141 (2015). ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00141.

[RHM+86] David E Rumelhart, Geoffrey E Hinton, James L McClelland, et al. "A general framework for parallel distributed processing". In: *Parallel distributed processing* 1.2 (1986).

[RHW86] David E Rumelhart, Geoffrey E Hintont, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536.

[RM10] Supratim Ray and J.H.R Maunsell. "Differences in gamma frequencies across visual cortex restrict their possible use in computation". In: *Neuron* 67.5 (2010), pp. 885–896.

[RM86] D.E. Rumelhart and J.L. McClelland. *Parallel distributed processing: explorations in the microstructure of cognition. Volume 1. Foundations*. Cambridge, MA, USA: MIT Press, 1986.

[RR11] Israel Rebollo Ruiz and Manuel Graña Romay. "Gravitational swarm approach for graph coloring". In: *Nature Inspired Cooperative Strategies for Optimization (NICSO 2011)*. Springer, 2011, pp. 159–168.

[RT87] Prabhakar Raghavan and Clark D Tompson. "Randomized rounding: a technique for provably good algorithms and algorithmic proofs". In: *Combinatorica* 7.4 (1987), pp. 365–374.

[Sch15] J. Schmidhuber. "Deep Learning in Neural Networks: An Overview". In: *Neural Networks* 61 (2015), pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003.

[Sch99] Uwe Schöning. "A probabilistic algorithm for k-SAT and constraint satisfaction problems". In: *Foundations of Computer Science, 1999. 40th Annual Symposium on*. IEEE. 1999, pp. 410–414.

[SGY13] D Sculley, Daniel Golovin, and Michael Young. "Big Learning with Little RAM". In: (2013).

[Sip96] M. Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1996.

[SMKGS13] Rupesh K Srivastava, Jonathan Masci, Sohrob Kazerounian, Faustino Gomez, and Jürgen Schmidhuber. "Compete to compute". In: *Advances in Neural Information Processing Systems*. 2013, pp. 2310–2318.

[Smo86] Paul Smolensky. "Information processing in dynamical systems: Foundations of harmony theory". In: (1986).

[SNPGFL15] Evangelos Stromatias, Daniel Neil, Michael Pfeiffer, Francesco Galluppi, Steve B Furber, and Shih-Chii Liu. "Robustness of spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms". In: *Frontiers in Neuroscience* 9.222 (2015).

[Sob67] Ilya M Sobol. "On the distribution of points in a cube and the approximate evaluation of integrals". In: *USSR Computational mathematics and mathematical physics* 7 (1967), pp. 86–112.

[Som87] H Sompolinsky. "The theory of neural networks: The Hebb rule and beyond". In: *Heidelberg colloquium on glassy dynamics*. Springer. 1987, pp. 485–527.

[SS00] Astrid Von Stein and Johannes Sarnthein. "Different frequencies for different scales of cortical integration: from local gamma to long range alpha/theta synchronization". In: *International Journal of Psychophysiology* 38.3 (2000), pp. 301–313.

[Ste13] Fabio Stefanini. "Robust online learning in neuromorphic systems with spike-based, distributed synapses". PhD thesis. Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 21337, 2013, 2013.

[STN01] P.J. Sjöström, G.G. Turrigiano, and S.B. Nelson. “Rate, Timing, and Cooperativity Jointly Determine Cortical Synaptic Plasticity”. In: *Neuron* 32.6 (2001), pp. 1149–1164.

[SZYD09] V. S. Sohal, F. Zhang, O. Yizhar, and K. Deisseroth. “Parvalbumin neurons and gamma rhythms enhance cortical circuit performance”. In: *Nature* 459.7247 (2009), pp. 698–702.

[Tie08] Tijmen Tieleman. “Training restricted Boltzmann machines using approximations to the likelihood gradient”. In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 1064–1071.

[Tie94] Luke Tierney. “Markov chains for exploring posterior distributions”. In: *the Annals of Statistics* (1994), pp. 1701–1728.

[Wal78] Peter Walker. “Binocular rivalry: central or peripheral selective processes?” In: *Psychological Bulletin* 85.2 (1978), p. 376.

[WHSW07] K.-F. Wong, A.C. Huk, M.N. Shadlen, and X.-J. Wang. “Neural circuit dynamics underlying accumulation of time-varying evidence during perceptual decision making”. In: *Frontiers in Computational Neuroscience* 1 (2007), pp. 6–.

[WSOSDEF07] Thilo Womelsdorf, Jan-Mathijs Schoffelen, Robert Oostenveld, Wolf Singer, Robert Desimone, A.K. Engel, and Pascal Fries. “Modulation of neuronal interactions through neuronal synchronization”. In: *science* 316.5831 (2007), pp. 1609–1612.

[YHCSSD14] Daniel LK Yamins, Ha Hong, Charles F Cadieu, Ethan A Solomon, Darren Seibert, and James J DiCarlo. “Performance-optimized hierarchical models predict neural responses in higher visual cortex”. In: *Proceedings of the National Academy of Sciences* (2014), p. 201403112.

# Lorenz Kaspar Müller

## PERSONAL DETAILS


| | |
|---|---|
| *Birth* | January 31, 1987 |
| *Origin* | Winterthur, ZH |
| *Address* | Heerenschürlistrasse 4c, 8051 Zurich |
| *Phone* | +41 79 414 01 61 |
| *Mail* | `lorenz.k.mueller@gmail.com` |


## EDUCATION

**PhD Neuroinformatics** 2011-2015

*University of Zurich*

Thesis on "Algorithms for Massively Parallel, Event-Based Hardware". I developed variants of artificial neural network algorithms for ultra low-power custom ASICs as well as a boolean satisfiablity solver for the same platform. The latter is the first application for which a neuromorphic platform can plausibly outperform a standard computer in terms of speed.

**MSc Physcis** 2009-2010

*ETH Zurich*

Grade average 5.55 of 6. Major in theoretical physics. Master's thesis in computational astrophysics on "Ultra Large-Scale Structures on the Light Cone".

**BSc Physcis** 2005-2009

*ETH Zurich*

Grade average 4.7 of 6.

**Swiss Highschool Diploma** 1999-2005

*Kantonsschule Rychenberg*

Grade average 5.6 of 6. Major in Ancient Languages (Latin and Greek). Bilingual courses in German and English.

## WORK EXPERIENCE

**PhD Researcher & Teaching Assistant** 2011-2015

*University of Zurich and ETH Zurich*

I researched the topics of my PhD thesis, supervised lab courses in physics and taught tutorial classes on neuroinformatics.

**Research & Teaching Assistant** 2009-2010

*ETH Zurich*

Co-wrote the script for the "Introduction to Computational Physics" class and later continued development of my master's thesis work on large-scale astrophysics simulations.

## SKILLS

| | |
|---|---|
| *Languages* | German (mother tongue), English (fluent), French (intermediate) |
| *Programming* | Python, C++, parallel programming in CUDA and MPI |